VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Storyboard Editor

Storyboard Editor

2017

Ondřej Horáček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Ondřej Horáček**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Storyboard Editor

Storyboard Editor

Jazyk vypracování: čeština

Zásady pro vypracování:

Práce bezprostředně navazuje na semestrální projekt Storyboard Editor. Cílem práce je pokračování ve vývoji Storyboard Editoru se zaměřením především na editaci vztahů mezi artefakty a na jejich hierarchickou strukturu vyjadřující princip dědičnosti. Součástí práce bude také vytvoření základní knihovny artefaktů a jejich využití na několika příkladech. Dále bude práce umožňovat export vztahů a faktů v textové podobě a ve formátu TIL script.
Editor umožní:
1. Editovat vztahy mezi jednotlivými artefakty.
2. Zachycení vztahů hierarchické struktury vyjadřující princip dědičnosti.
3. Export vztahů a faktů v textovém formátu a ve formátu TIL script.
Práce bude obsahovat:
1. Popis využitých technologií.
2. Implementaci výše zmíněné funkcionality.
3. Programátorskou dokumentaci řešení s využitím diagramů jazyka UML.
4. Uživatelskou dokumentaci.

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] Jezek, D. & Stolfa, S.. STORYBOARDS METHOD AND STRUCTURED SHOT. Novais, P.; Machado, J.; Analide, C. & Abelha, A.(Eds.). EUROSIS, 2011., 99-101 ISBN 978-90-77381-66-3 EUROPEAN SIMULATION AND MODELLING CONFERENCE 2011, EUROSIS, 2011, 99-101

Dále dle pokynů vedoucího práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28. dubna 2017 Horáček .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 28. dubna 2017 Horáček .....

Rád bych na tomto místě poděkoval vedoucímu mé diplomové práce Ing. Davidu Ježkovi, Ph.D., za veškeré cenné rady, nápady a poznatky na vylepšení projektu. Dále bych rád poděkoval Václavovi Vaňkovi a Adamovi Zikmundovi za spolupráci při vývoji v rámci semestrálního projektu.

Abstrakt

Cílem této diplomové práce je seznámení čtenáře s alternativní metodou pro modelování podnikových procesů pomocí storyboardů a její implementací v prostředí Eclipse Rich Client Platform (RCP). V první části je čtenáři představena metoda storyboardů a základní struktura vytvářených storyboardů. Druhá část práce se poté zabývá popisem technologií a nástrojů využívaných při vývoji aplikace a popisem vytváření Eclipse RCP aplikací a jejich úpravami. Poslední část je nakonec věnována implementaci řešení.

Klíčová slova: Storyboard, Editor, Eclipse, RCP, Java, diplomová práce

Abstract

Objective of this Thesis is to acquaint readers with alternative method of business process modeling using storyboards and its implementation in Eclipse Rich Client Platform (RCP). In the first part the reader is introduced to storyboard method and its basic structure of created storyboards. The second part then describes technologies and tools used during application development and also describes creation of Eclipse RCP application and their modifications. The last part is devoted to solution implementation.

Key Words: Storyboard, Editor, Eclipse, RCP, Java, master thesis

Obsah

Seznam použitých zkratek a symbolů **10**

Seznam obrázků **11**

Seznam tabulek **13**

Seznam výpisů zdrojového kódu **14**

1 Úvod **15**

2 Úvod do storyboardů **16**

2.1 Proces . . . 16
2.2 Podnikový (byznys) proces . . . 16
2.3 Byznys modelování . . . 16
2.4 Storyboard . . . 16
2.5 Struktura storyboardu . . . 17

3 Využívané technologie a nástroje **19**

3.1 Správa artefaktů . . . 19
3.2 Správa zdrojových kódů . . . 19
3.3 Správa diagramů a wireframů . . . 22
3.4 Eclipse . . . 22
3.5 Eclipse RCP . . . 23
3.6 Sestavování projektu . . . 25

4 Implementace **32**

4.1 Rozšířený model struktury storyboardu . . . 32
4.2 Rozdělení balíčků . . . 33
4.3 Service - ukládání a načítání . . . 33
4.4 Práce s SVG . . . 36
4.5 Modelování, vykreslování objektů . . . 38
4.6 Popis částí aplikace . . . 42

5 Závěr **59**

Literatura **60**

Přílohy **60**

A Uživatelská dokumentace **61**
A.1 Popis aplikace 61
A.2 Spuštění aplikace 61
A.3 Vytváření projektů, procesů, scén 61
A.4 Změna vlastností projektů, procesů, scén 62
A.5 Sestavování scén 63
A.6 Vytváření a úprava figure 67
A.7 Aktualizace figure 68
A.8 Změna adresáře figures 69

B CD s aplikací **70**

Seznam použitých zkratek a symbolů

RCP	–	Rich Client Platform
UML	–	Unified Modeling Language
SVG	–	Scalable Vector Graphics
XML	–	eXtensible Markup Language
URI	–	Uniform Resource Identifier
TIL	–	Transparent Intensional Logic

Seznam obrázků

1 Ukázka storyboardu . . . 17
2 Základní model struktury projektu . . . 18
3 Distribuované verzovací systémy . . . 20
4 Vrstvy Gitu . . . 21
5 Graf commitů na projektu . . . 22
6 Aplikace Creately . . . 22
7 Modelové elementy - Okno . . . 23
8 Modelové elementy - Díl . . . 24
9 Modelové elementy - kontejner . . . 24
10 Hierarchie kontejnerů . . . 25
11 Základní aplikace po sestavení projektu pomocí Gradle Wuff . . . 27
12 Přidání dílu do aplikace . . . 28
13 Vytvoření třídy k dílu . . . 28
14 Vytvoření handleru pro menu v aplikaci . . . 30
15 Vytvoření položky menu . . . 31
16 Rozšířený model struktury tříd . . . 32
17 Projektová struktura . . . 36
18 Sekvenční diagram převodu SVG obrázku . . . 38
19 Diagram tříd balíčku modeling . . . 39
20 Ukázka aplikace a rozdělení na části . . . 42
21 Diagram tříd pro část Project structure . . . 43
22 Zobrazení projektové struktury v aplikaci . . . 44
23 Ukázka exportu do textového formátu . . . 45
24 Ukázka exportu do formátu TIL Script . . . 45
25 Diagram tříd pro část Properties . . . 48
26 Diagram tříd pro část Active area . . . 50
27 Diagram aktivit pro první případ . . . 52
28 Diagram aktivit pro druhý případ . . . 53
29 Diagram tříd pro část Figure list . . . 54
30 Ukázka vzhledu editoru . . . 57
31 Hlavní okno aplikace . . . 61
32 Vytváření nové scény . . . 62
33 Zobrazení vlastností vybraného objektu . . . 63
34 Načtení procesu na Active area . . . 64
35 Menu pro figure umístěný na scéně . . . 64
36 Okno pro úpravu figure umístěných na scéně - menu Properties . . . 65
37 Označení figure a následně i aktivní oblasti ve figure . . . 66

38 Vytvoření connection na scéně 66
39 Editor pro vytváření a úpravu figure 67
40 Dialog detekující změny figure 68
41 Dialog pro zvolení aktualizovaných figure 68
42 Dialog oznamující smazané connections 69
43 Okno Preferences pro změnu umístění adresáře knihovny figures 69

Seznam tabulek

1 Rozdělení správy artefaktů 20

Seznam výpisů zdrojového kódu

1 Výchozí obsah build.gradle . . . 26
2 Nastavení parametru products . . . 26
3 Nastavení parametru dependencies . . . 27
4 Ukázka vygenerované třídy tvořící Part . . . 29
5 Ukázka vygenerovaného handleru . . . 30
6 Ukázka využití notací @XmlElements a @XmlElementWrapper . . . 34
7 Ukázka uložení objektu pomocí JAXB . . . 34
8 Ukázka načtení objektu pomocí JAXB . . . 35
9 Ukázka uloženého figure . . . 35
10 Vytvoření task pro SVG Salamander . . . 37
11 Využití třídy Tracker . . . 40
12 Nastavení providerů pro TreeViewer . . . 44
13 Zasílání eventů v aplikaci . . . 46
14 Odchytávání eventů v aplikaci . . . 46
15 Ukázka implementace metody getPropertyItems() . . . 48
16 Ukázka implementace metody setValue(...) . . . 49
17 Vytváření DragSource . . . 55
18 Vytváření DropTarget . . . 55
19 Nastavení defaultních hodnot . . . 57

1 Úvod

Účinný popis podnikových procesů je jedním z nejdůležitějších úkolů, kterého je potřeba docílit k dosažení efektivity firmy. Při popisu procesů zainteresovaným osobám ovšem dochází k problému, když dané osobě chybí znalosti pro pochopení formálních notací a definic podnikových procesů. Proto je potřeba metoda byznys modelování, která je snadno pochopitelná pro každou zainteresovanou osobu a která má zároveň formální základ pro analytiky. Jednou z takových metod je metoda storyboard, na které staví tato práce.

Aplikace Storyboard Editor byla vyvíjena již v rámci semestrálního projektu, kde se na vývoji podíleli další dva členové týmu, a následně jsem pokračoval už sám úpravami a rozšiřováním možností editoru v rámci této diplomové práce.

Textová část práce je rozdělena na 3 hlavní kapitoly. První kapitola se snaží objasnit čtenáři problematiku modelování podnikových procesů. Popisuje základní pojmy týkající se modelování podnikových procesů. Následně seznamuje čtenáře s pojmem storyboard, popisuje využití a cíle metody storyboard a také strukturu storyboardů. Nakonec tato kapitola obsahuje i základní strukturu, se kterou se storyboard editor následně vyvíjel.

Druhá kapitola se poté věnuje technologiím a nástrojům využívaných při vývoji projektu. První část této kapitoly popisuje nástroje využívané pro správu jednotlivých artefaktů projektu. Druhá část kapitoly se poté zabývá prostředím Eclipse RCP a popisuje modelové elementy uživatelského rozhraní. Dále je v kapitole popsán způsob vytváření vlastní Eclipse RCP aplikace pomocí Gradle Wuff pluginu, který byl využit při vytváření Storyboard editoru. Nakonec je v kapitole popsána i další práce s takto vytvořenou aplikací, přidávání komponent a menu do RCP aplikace.

Třetí kapitola poté obsahuje již samotnou implementaci Storyboard editoru. Na začátku je představen a popsán rozšířený model storyboardu. Dále jsou popsány jednotlivé balíčky v projektu a jejich využití. Následně je popsán způsob práce s daty, jakým stylem jsou data načítány a ukládány. Poté je v kapitole popsán způsob modelování a vykreslování objektů a dále je popsána struktura jednotlivých částí aplikace, způsob řešení daných částí a komunikace mezi nimi.

2 Úvod do storyboardů

Pro jednodušší pochopení toho, co si lze představit pod pojmem storyboard, je dobré znát některé pojmy, které se budou při popisu využívat. Mezi tyto pojmy patří např. proces, podnikový (byznys) proces a modelování procesů. Proto tyto pojmy nejprve trochu popíšu.

2.1 Proces

Lze najít spousta definicí pro proces, které jsou napsány různě, ale hlavní myšlenka je stejná. Podle normy ISO 9001 je proces definován jako "Soubor vzájemně působících činností, který přeměňuje vstupy na výstupy."[1]

2.2 Podnikový (byznys) proces

Podnikový proces říká, jak se má postupovat při vykonávání určitého úkolu, popř. provádění specifické akce. Podnikový proces je tedy posloupností na sebe navazujících aktivit nebo úkolů, jejichž výsledkem je vytvoření nějakého produktu nebo vykonání určité služby, kterou zákazník vyžaduje. Tyto procesy mohou být často popisovány pomocí vývojových diagramů, kde jednotlivé obrazce reprezentují kroky procesu a šipky směr toku řízení.

2.3 Byznys modelování

Cílem byznys modelování je snaha zachytit současný stav nějakého podniku, případně i předpokládaný budoucí stav, kterého chce podnik dosáhnout. K tomu je často využíváno modelování podnikových procesů. Tyto modely mohou následně sloužit jak k popisu daného procesu, tak i k analýze procesu a případně k reengineeringu podnikového procesu. Pod pojmem reengineering si můžeme představit nějaké zásadní přehodnocení a rekonstrukci procesů tak, aby bylo dosaženo zlepšení z hlediska měření výkonnosti, jako jsou např. náklady, kvalita, rychlost.

Pro popis procesů lze využít UML diagramy. Ty jsou vhodné pro tento popis procesu, když se snažíme například vysvětlit zaměstnanci v podniku, jak daný proces funguje. Zatímco zaměstnanci, případně analytici, mohou rozumět vytvořeným modelům a UML diagramům, které jim stačí pro popis procesů, nastanou i situace, kdy je potřeba prezentovat nějaké podnikové procesy i zákazníkům, akcionářům a dalším zainteresovaným osobám. Ovšem zde může nastat problém. Co když tento zákazník nerozumí danému popisu procesu, jelikož se nevyzná v UML syntaxi, atd. V takových případech by bylo vhodné reprezentovat popis procesu takovým způsobem, ke kterému nejsou potřeba nějaké technické znalosti. A pro tento popis lze využít storyboardy.

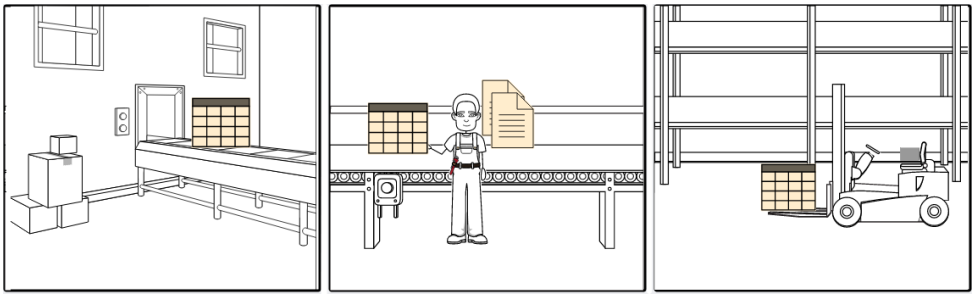
2.4 Storyboard

Cílem metody storyboardu je zjednodušení a zobrazení popisu procesů takovým způsobem, kterému bude rozumět každý, kdo alespoň trochu chápe účel daného procesu. Storyboardy také

umožňují následnou transformaci určitých informací do formální podoby, která může být základem pro vytváření formálního popisu procesu nebo modelu pro simulaci.

Storyboardy jsou užitečným nástrojem pro plánování projektů, které vykreslují kroky, ze kterých se projekt skládá. Poskytuje náhled na vše, co se děje v každém kroku, a kombinace různých různých storyboardů poskytuje detailní náhled na proces. Storyboardy odhalují chybějící úkoly, problémy a komplikace, které se mohou vyskytnout. To umožňuje zavčasu provést změny a přerozdělit úkoly podle potřeby dříve, než se proces začne využívat v praxi.[2]

Storyboard je grafickou interpretací průběhu daného procesu. Skládá se ze sekvence vzájemně na sebe navazujících snímků, kde každý ze snímků reprezentuje určitý fragment popisovaného procesu, tedy určitou akci (viz obrázek 1). Každá tato sekvence snímků znázorňuje specifický průchod tímto procesem. Každý snímek se skládá z ilustračního obrázku určité aktivity, jeho jména, popisku a dalších atributů.



Obrázek 1: Ukázka storyboardu

2.5 Struktura storyboardu

Pro realizaci storyboardů je vhodné si nejprve představit, jak by taková struktura storyboardů mohla vypadat. V mém případě jsem se rozhodl seskládat strukturu storyboardů ze čtyř úrovní (viz obrázek 2).

2.5.1 Projekt

Nejvyšší úroveň tvoří projekty. Ty umožňují rozdělení procesů do určitých skupin. Např. projekt Skladiště bude obsahovat všechny popisované procesy, které se týkají práce ve skladišti.

2.5.2 Proces

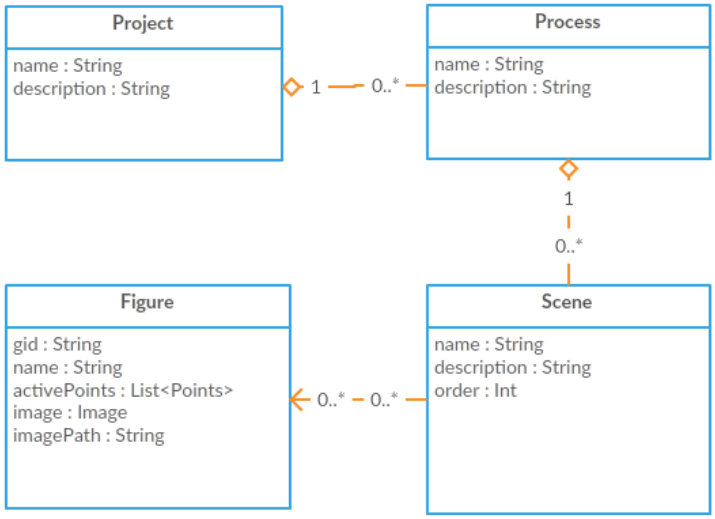
Každý projekt se tedy skládá ze seznamu procesů reprezentujících reálné podnikové procesy, pro které vytváříme storyboardy. Jak již bylo zmíněno, storyboardy jsou sekvence snímků popisujících proces, proto je v každém procesu uložen seznam snímků tvořících daný storyboard.

2.5.3 Scéna

Scény reprezentují jednotlivé snímky, tedy určitou část popisovaného procesu. Pokud by šlo pouze o zobrazení storyboardů, tyto tři úrovně by mohly stačit. Ovšem jelikož potřebujeme i vytvářet scény, je zapotřebí jít ještě více do detailu až na jednotlivé prvky, ze kterých se scéna skládá.

2.5.4 Figure

Figure reprezentují jednotlivé artefakty, ze kterých je scéna tvořena. Figure tedy mohou být na dané scéně např. dokumenty, osoby, vozidla a další.



Obrázek 2: Základní model struktury projektu

3 Využívané technologie a nástroje

3.1 Správa artefaktů

Snad v každém projektu dojde na okamžik, kdy je potřeba určit, jakým způsobem budou spravovány jednotlivé artefakty, které se v rámci vývoje projektu vytváří. Je důležité uchovávat si historii, jednotlivé prováděné změny v projektu, aby bylo možné se v případě potřeby vrátit zpět k specifické předešlé verzi.

V tomto projektu se mezi artefakty řadí:

1. zdrojové kódy
2. diagramy
3. testovací data
4. inicializační data

Pro správu těchto artefaktů je důležité, aby již na začátku vývoje byly stanoveny pravidla určující způsob správy tak, aby při práci na projektu tyto věci již fungovaly automaticky a nezdržovaly.

Pro tento projekt bylo použito následující rozdělení správy jednotlivých artefaktů (viz tabulka 1).

3.2 Správa zdrojových kódů

Pro správu zdrojových kódů byl zadáním určen systém GIT. Předtím, než zde napíšu něco o systému GIT, bylo by vhodné rozumět i pojmu verzování, proto bude následující text věnován nejprve tomuto pojmu.

3.2.1 Verzování

Verzování je systém zaznamenávající změny v souborech v průběhu času tak, aby se k nim bylo možné později vrátit. Umožňuje nahlédnout na změny, zjistit, kdo například provedl změnu, která byla příčinou nějakého problému, případně umožňuje jednoduše návrat k poslední funkční revizi.

Mnoho lidí provádí verzování stylem, že soubory překopírují do jiné složky. Tato možnost je sice jednoduchá, ale dost náchylná na chyby (nechtěné přepsání souborů, změny do špatného souboru, atd.). Z tohoto důvodu byly vyvinuty lokální verzovací systémy, které měly jednoduchou databázi uchovávající všechny změny souborů pod jednotlivými revizemi.

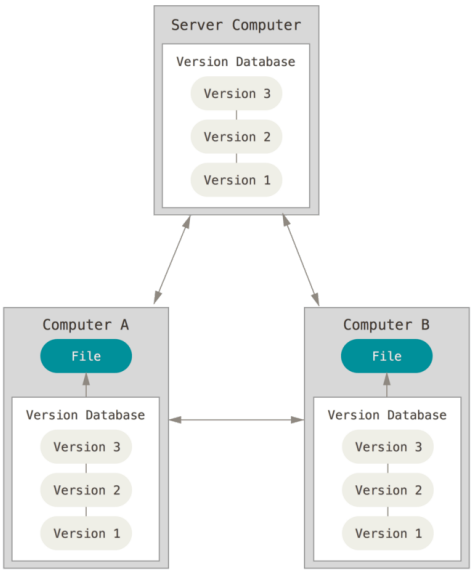
Ovšem vyskytl se problém, kdy byla potřeba i spolupráce více vývojářů. Řešením bylo vyvinutí centralizovaných verzovacích systémů. Ty měly jediný server, který obsahoval všechny verzované soubory. Oproti lokálním verzovacím systémům měly centralizované systémy mnoho

Tabulka 1: Rozdělení správy artefaktů

Artefakt	Způsob správy
zdrojové kódy	git
diagramy	creately
wireframy	creately
testovací data	git
inicializační data	git

výhod. Každý mohl do určité míry vidět, co dělají na projektu ostatní. Také administrativa jednoho serveru byla jednodušší než administrativa každé lokální databáze. Ovšem výskyt jednoho serveru byl i nevýhodou, jelikož pokud došlo například k chybě na serveru, který musel být na určitou dobu odstaven, nemohli v té době vývojáři vůbec spolupracovat nebo ukládat verzované změny jejich práce. Navíc pokud došlo ke zničení disku, na kterém je centrální databáze uložena, a nebyla udržována záloha, došlo ke ztrátě veškeré historie projektu.

Zde nastoupily distribuované verzovací systémy (DVCS). Zde uživatel nemá u sebe uloženou pouze poslední verzi souborů, ale má nakopírovaný celý repozitář. Díky tomu, pokud by došlo k poškození serveru, může být repozitář libovolného klienta nakopírován zpět na server pro jeho obnovení.



Obrázek 3: Distribuované verzovací systémy

3.2.2 Git

Mezi distribuované verzovací systémy patří i Git, který byl použit v tomto projektu.

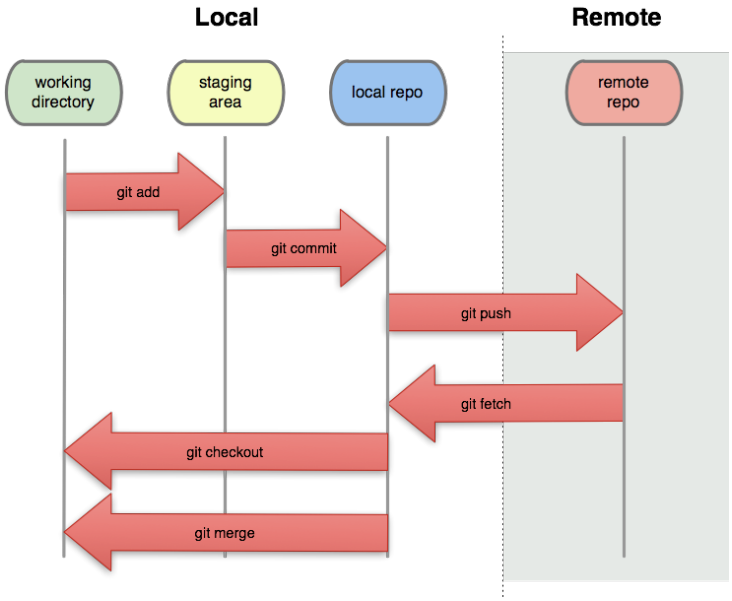
Git byl vyvíjen s cílem vytvořit plně distribuovaný, rychlý systém s jednoduchým návrhem, podporou nelineárního vývoje (pomocí větvení - branching), který by byl i schopný zvládat velké

projekty účinně. Od jeho vyvinutí v roce 2005 se dále rozvíjel do podoby, kdy je jednoduchý na použití a stále se drží původních cílů. Je rychlý, výkonný a podporuje nelineární vývoj.

Většina operací v Gitu využívá pouze lokální soubory a zdroje. To znamená, že například když chceme vyhledat historii projektu, není potřeba připojit se na server a načíst a zobrazit historii odtud. Stačí ji přečíst přímo z lokální databáze, čímž ji dostaneme téměř okamžitě. Také to znamená, že nemusíme být připojeni k síti, abychom mohli pracovat. Můžeme commitovat změny stále a ty nasdílet poté, co se připojíme k síti.

Git obsahuje tři hlavní stavy, ve kterých mohou soubory být. Commited pro data, která jsou uloženy bezpečně v lokální databázi, modified pro soubory, které byly upraveny, ale ještě nebyly commitnuty do databáze, a staged stav pro upravené soubory označené v současné verzi, které budou uloženy v dalším commitu.

To vede ke třem vrstvám, ve kterých Git pracuje - Working directory, Staging area, Local repository (viz obrázek 4).



Obrázek 4: Vrstvy Gitu

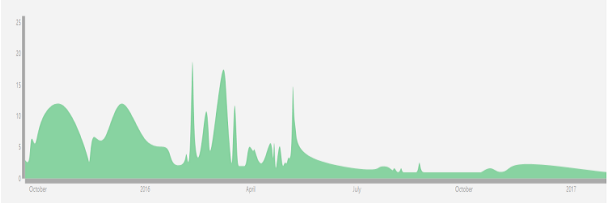
Working directory je naše složka, ve které pracujeme. Staging area slouží pro soubory, které chceme, aby Git zpracoval. Je to v podstatě virtuální složka pro Git, kde to, co se do ní dostane, bude příštím commitem přeneseno do lokálního repozitáře. Lokální repozitář je naše historie projektu, na kterém pracujeme. Uchovává snapshoty a všechny předchozí verze projektu. Ten poté nahráváme na vzdálený repozitář.[4]

3.2.3 GitLab

V průběhu vývoje projektu byl Git repozitář přesunut do systému GitLab. GitLab je webový manažer Git repozitářů, podobný službě GitHub. Kromě správy repozitářů umí GitLab zejména

také Wiki a Issue tracking. Díky systému Gitlab jsem získal možnost si prostřednictvím webového prohlížeče prohlížet upravené soubory, jednotlivé commity apod. Také byla otevřena cesta pro jednodušší integraci s dalšími systémy pro podporu vývoje.

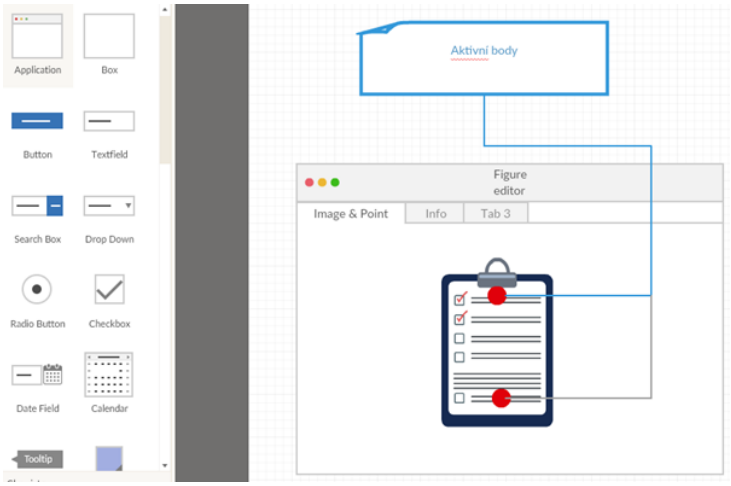
Systém Gitlab generuje přehledné statistiky commitů, které agreguje přes email commitera. Ke dni 17.04.2017 bylo provedeno celkem 413 commitů.



Obrázek 5: Graf commitů na projektu

3.3 Správa diagramů a wireframů

Existuje velké množství aplikací na správu, vytváření a editaci diagramů. Nakonec jsme se v týmu rozhodli pro cloudové řešení Creately (viz obrázek 6). Aplikace je dostupná skrz webový prohlížeč. Stejná aplikace byla využita i pro tvorbu wireframe, protože tato aplikace poskytuje nepřeberné množství hotových GUI komponent, které jsou ideální pro rychlé prototypování a navrhování obrazovek.



Obrázek 6: Aplikace Creately

3.4 Eclipse

Eclipse je integrované vývojové prostředí používané pro programování převážně v jazyce Java. Umožňuje však i rozšiřitelnost funkcionality pomocí pluginů, například o jazyky C, C++ nebo PHP. Dále lze také pomocí pluginů dosáhnout i možnosti vytvářet UML diagramy, zapisovat HTML či XML soubory. První nasazení Eclipse proběhlo již před 15 lety 7. listopadu 2001 s

verzí 1.0. Od té doby je pravidelně Eclipse aktualizován a v současné době běží na verzi 4.6 (Neon) a v letošním roce by měla vyjít i verze 4.7 (Oxygen).

3.5 Eclipse RCP

Eclipse RCP je zkratkou pro Eclipse Rich Client Platform a naznačuje, že Eclipse platforma je používána jako základ pro snadnější vytváření graficky bohatých aplikací. Eclipse RCP aplikace využívají existující uživatelské rozhraní a vnitřní rámce a umožňují opětovné použití pluginů a funkcí. Tato platforma využívá knihovny SWT a JFace.

Pojem Eclipse RCP byl vytvořen v roce 2004, kdy byla vydána verze Eclipse 3.0. Ta podporovala opětovné použití platformy Eclipse na sestavení samostatných aplikací založených na stejné technologii jako Eclipse IDE.[5]

Proč si vybrat Eclipse RCP?

1. Platforma Eclipse tvoří základ spousty úspěšných Java IDE a proto je velmi stabilní a široce využívána.

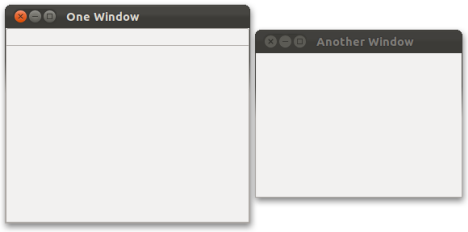
2. V původním nastavení poskytuje nativní komponenty uživatelského rozhraní, které jsou rychlé a spolehlivé.

3. Společnosti jako IBM, SAP a Google používají rámec Eclipse jako základ pro jejich produkty a proto zajišťují, že je Eclipse flexibilní, rychlý a stále se rozvíjí.

3.5.1 Modelové elementy uživatelského rozhraní

3.5.1.1 Okno

Eclipse aplikace se skládají z jednoho nebo více oken. Typicky mívá aplikace pouze jedno okno, ale není to na tento počet omezeno.



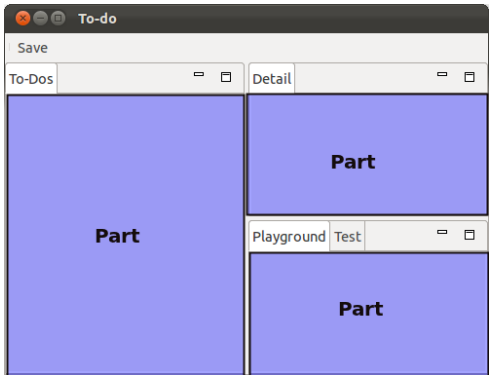
Obrázek 7: Modelové elementy - Okno

3.5.1.2 Díl

Díly jsou komponenty uživatelského rozhraní umožňující navigaci a modifikování dat. Mohou být seskládány vedle sebe nebo naskládány v závislosti na kontejneru, do kterého jsou vloženy.

Díly mohou být klasifikovány jako pohledy a editory. Pohled je typicky použit pro práci s daty, které mohou mít hierarchickou strukturu. Pokud jsou data v pohledu změněny, změna bývá

aplikována na danou datovou strukturu. Editory jsou typicky použity pro modifikování jednoho datového elementu jako např. obsah souboru nebo datového objektu.



Obrázek 8: Modelové elementy - Díl

3.5.1.3 Kontejnery dílů

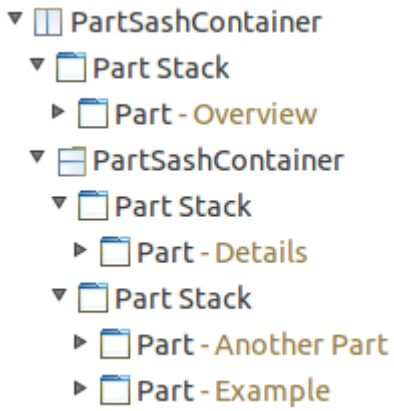
Díly mohou být přiřazeny buď do okna, nebo do perspektivy. Mohou být také seskupeny a organizovány pomocí dodatečných modelových elementů. Tyto elementy jsou buď zásobník dílů (Part Stack) nebo posuvný kontejner dílů (Part Sash Container).

Zásobník dílů obsahuje skupinu dílů, kdy zobrazuje vždy jen jeden díl a hlavičky dalších dílů v záložkách. Uživatel poté může přepínat mezi zobrazovanými díly tak, že si vždy vybere příslušnou záložku.

Posuvný kontejner dílů zobrazuje všechny své díly současně, vždy uspořádané buď horizontálně nebo vertikálně.

Obrázek 9: Modelové elementy - kontejner

Na obrázku 9 lze vidět strukturu, kdy nejvyšší vrstvu tvoří posuvný kontejner dílů s horizontálním zarovnáním. Ten obsahuje zásobník dílů obsahující pouze jeden díl, a to Overview. Dále poté kontejner obsahuje další posuvný kontejner dílů, který má nyní vertikální zarovnání a obsahuje dva zásobníky dílů. První z nich má opět pouze jeden díl (Details) a druhý zásobník obsahuje dva díly. Another Part a Example.



Obrázek 10: Hierarchie kontejnerů

3.5.1.4 Perspektiva

Perspektiva je volitelný kontejner pro sadu dílů. Perspektiva může uchovávat různé uspořádání jednotlivých dílů. Například v Eclipse IDE jsou používány pro vhodné rozložení pohledů v závislosti na úkolu (development, debugging, atd.), který chce vývojář vykonat.

3.6 Sestavování projektu

Sestavování projektu je věc, která je často řešena až, když je potřeba. U tohoto projektu jsme se v týmu rozhodli řešit tuto věc hned na začátku projektu. Vzhledem k zadané vývojové platformě Eclipse RCP jsme měli na výběr mezi:

1. standardní eclipse plugin
2. maven - tycho plugin
3. gradle - wuff plugin

Po zkušenostech z příprav na tento projekt a neautomatické správě závislostí a složitému nasazení aplikace jsme první možnost zamítli. Druhou možnost (maven tycho) jsme zkoušeli, ale po počátečních neúspěších jsme zvolili implementačně jednodušší a přívětivější gradle wuff plugin.

3.6.1 Gradle

Gradle je nástroj pro automatizaci sestavování programu, který staví na zkušenostech s nástroji Apache Ant a Apache Maven. Pro deklarování konfigurace projektu se využívá doménově specifický jazyk (DSL) namísto XML, který používá Apache Maven. Původní pluginy jsou zaměřeny primárně kolem Java, Groovy a Scala jazyků.[6]

3.6.2 Gradle Wuff

Wuff je gradle plugin pro vývoj a sestavení OSGi/Eclipse aplikací a pluginů nezávisle na Eclipse IDE.[7]

Pro vytvoření RCP aplikace pomocí wuff pluginu je potřeba mít nainstalovaný pouze gradle. V projektové složce se vytvoří soubor `build.gradle`, který popisuje sestavovanou aplikaci. Výchozím obsahem tohoto souboru je popis sestavované aplikace (viz výpis kódu 1.

```
buildscript {
  repositories {
    mavenLocal()
    jcenter()
  }

  dependencies {
    classpath 'org.akhikhl.wuff:wuff-plugin:+'
  }
}

apply plugin: 'java'
apply plugin: 'org.akhikhl.wuff.eclipse-rcp-app'

repositories {
  mavenLocal()
  jcenter()
}
```

Výpis 1: Výchozí obsah build.gradle

Pro vytvoření aplikace poté stačí spustit příkaz `gradle scaffold`, který stáhne a vytvoří všechny potřebné soubory. Následně lze pomocí příkazu gradle run spustit vytvořenou aplikaci.

Dále lze do `build.gradle` přidat parametr `products`, ve kterém lze nastavit, pro které platformy se má výsledná aplikace sestavovat (viz ukázka kódu 2. Pro sestavení aplikace je třeba následně spustit příkaz `gradle build`, který sestaví balíčky pro jednotlivé platformy ve složce `/build/output`.

```
products {
     product platform: 'windows', arch: 'x86_64'
     product platform: 'macosx', arch: 'x86_64'
     archiveProducts = true
}
```

Výpis 2: Nastavení parametru products

Jedním z primárních důvodů, proč bylo využito nějakého systému, který deklarativně popisuje sestavovanou aplikaci, je správa závislostí. V souboru `build.gradle` lze pomocí parametru `dependencies` nastavit závislosti podobně jako třeba v maven `pom.xml` souboru (viz ukázka kódu 3).

```
dependencies {
   compile fileTree(dir: 'libs', include: ['*.jar'])
   compile 'com.google.guava:guava18.0'
   compile 'commons-io:commons-io:2.4'
   testCompile 'junit:junit:4.12'
}
```

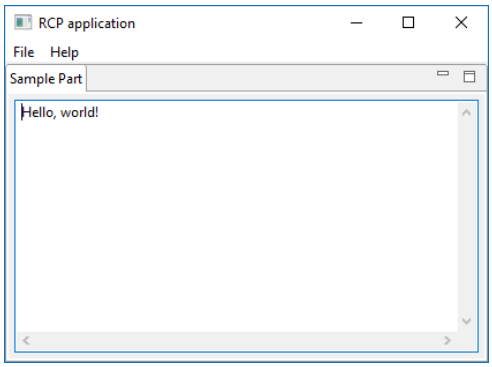
Výpis 3: Nastavení parametru dependencies

Bohužel tento systém není v podání wuff pluginu ještě dokonalý a má problém při sestavování složitějších závislostí.

Použití gradle namísto obyčejného RCP projektu umožňuje snazší integraci s jinými IDE, popřípadě s nástroji pro CI (Continuous Integration).

Pro použití v Eclipse IDE je třeba doinstalovat Eclipse Gradle plugin. Následně lze projekt naimportovat a skrz nabídku Run volat konkrétní gradle task (run, clean, build, ...).

Pro ladění stačí aplikaci spustit pomocí příkazu `gradle debug`. V tu chvíli aplikace čeká na připojení debuggeru na standardním portu 5005. Po připojení lze využít všechny možnosti Eclipse debuggeru.



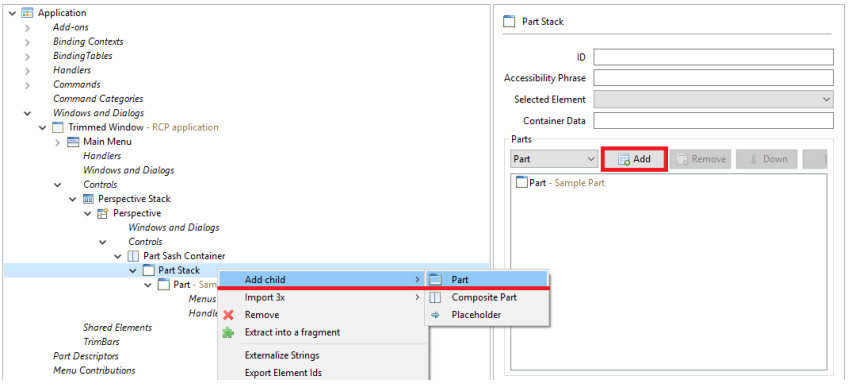
Obrázek 11: Základní aplikace po sestavení projektu pomocí Gradle Wuff

3.6.3 Přidávání dílů do aplikace

Přidávání a rozložení dílů v aplikaci se provádí v souboru `Application.e4xmi`. Kromě již zmíněných možností se zde přiřazují k jednotlivým dílům i třídy, které se starají následně o obsah daného dílu aplikace. Dále zde jdou nastavit například i položky a rozložení menu.

Základní aplikace vytvořená pomocí gradle wuff pluginu se skládá z jednoho okna obsahujícího několik položek menu a dále kontejneru Part Sash Container (viz obrázek 11), který se

dá využít jako základ pro rozmístění komponent. Tento kontejner již v této základní podobě obsahuje jeden Part Stack, ve kterém je umístěn jeden Part. Přidání nového dílu, případně dalšího kontejneru, je možné buď stiskem pravého tlačítka myši a zvolení možnosti Add child, nebo zvolením možnosti Add v okně pro nastavení vlastností objektu (viz obrázek 12).

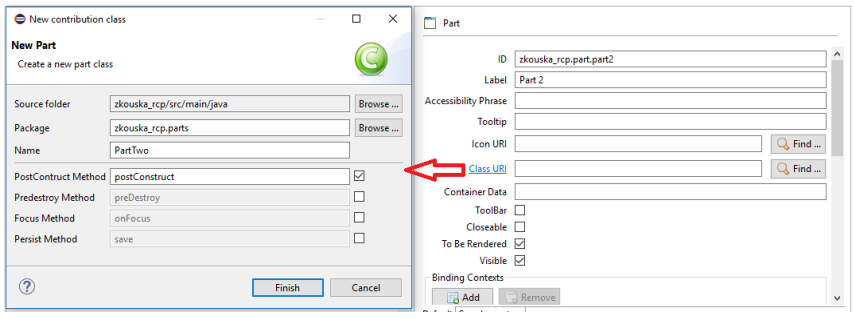


Obrázek 12: Přidání dílu do aplikace

Po provedení změn v souboru `Application.e4xmi` je potřeba nejprve provést `gradle clean`, aby byly při příštím spuštění příkazu `gradle run` změny viditelné. Nyní je vytvořen nový díl, který je již viditelný v aplikaci. Ovšem v tuto chvíli nemá tento nový díl žádnou funkcionalitu. Proto je potřeba mu přiřadit třídu, která se bude starat o obsah tohoto dílu. To se provede tak, že se danému dílu nastaví vlastnost Class URI. To můžeme udělat několika způsoby. Buď napíšeme přímo URI pro danou třídu, nebo třídu vyhledáme pomocí tlačítka "Find ...", případně kliknutím na Class URI se otevře okno pro vytvoření třídy (viz obrázek 13).

Class URI se skládá ze tří částí. První část tvoří prefix `"bundleclass://"`. Následuje symbolický název pro balík, který je ukončen lomítkem. Poslední část poté tvoří plně kvalifikovaný název třídy. Celý Class URI vypadá například takto:

`bundleclass://rcp_application/rcp_application.parts.PartTwo`



Obrázek 13: Vytvoření třídy k dílu

Pokud jsme třídu vytvářeli přes Class URI okno, vytvoří se nová třída, ve které se následně může psát obsah dílu. Tato vygenerovaná třída obsahuje konstruktor a hlavně metodu s ano-

tací `@PostConstruct` (viz ukázka kódu 4). V této metodě se následně tvoří obsah daného dílu aplikace.

```
public class PartTwo {
   @Inject
   public PartTwo() {

   }

   @PostConstruct
   public void postConstruct(Composite parent) {

   }
}
```

Výpis 4: Ukázka vygenerované třídy tvořící Part

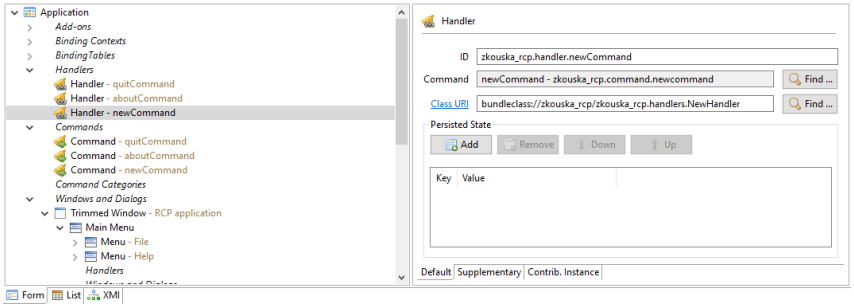
Jednotlivé kontejnery, zásobníky a díly mají možnost nastavení i vlastnosti containerData. Tou se nastavuje váha jednotlivých částí, tedy jakou část okna zabírá daná část. Po nastavení hodnoty containerData u jedné části je potřeba nastavit i hodnoty ostatních částí, jinak bude při spuštění aplikace tato část považována za dominantní a zabere celou plochu rodičovského kontejneru. Doporučuje se nastavovat hodnoty v řádech tisíců, např. 6000, atd., případně vyšší. Není doporučené používat malé hodnoty, jelikož poté může docházet při změně velikosti části v aplikaci ke skokové změně o několik pixelů.

3.6.4 Přidávání menu

Pomocí aplikačního modelu lze přidávat do RCP aplikace i menu a panely nástrojů. Ty mohou být umístěny na různá místa. Například můžeme přidat menu k oknu aplikace nebo i jen k nějaké části. Tyto elementy definují odkaz na třídu, kde instance této třídy je zodpovědná za chování, pokud položka menu nebo panelu nástrojů je zvolena. Taková třída je nazývána handler.

Aplikační model umožňuje specifikovat tzv. commands a handlers. Jejich používání je volitelné, ovšem doporučuje se. Command je deklarativní popis abstraktní akce, která může být provedena (např. edit, copy, save, atd.). Command je nezávislý na detailech implementace. Eclipse neposkytuje standardní příkazy, takže si je musíme vytvořit v aplikačním modelu sami. Chování commandu je následně definované pomocí handleru. Modelový element handleru podobně jako u Part obsahuje Class URI, které ukazuje na třídu starající se o tento handler.

Pro vytvoření menu položky je nejprve potřeba v `Application.e4xmi` vytvořit tedy Command, kterému se následně nastaví název. Následně se vytvoří Handler (viz obrázek 14), kterému se přiřadí daný Command a poté, podobně jako u vytváření Part, se mu nastaví Class URI, ve kterém bude popsána logika.



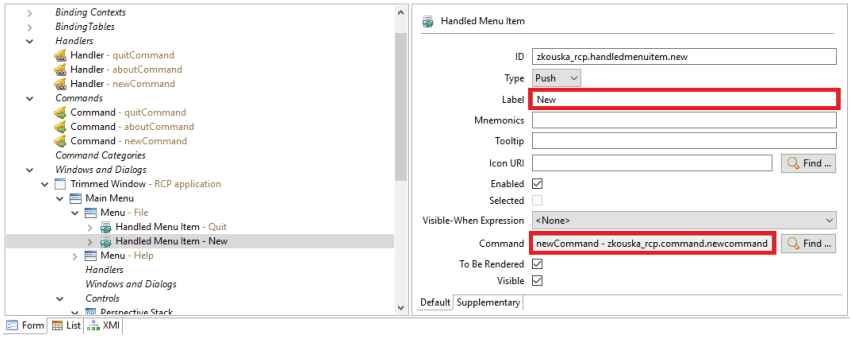
Obrázek 14: Vytvoření handleru pro menu v aplikaci

Pokud byla třída pro handler vygenerována přes Class URI, bude obsahovat metodu s anotací `@Execute`, což označuje metodu, která je zodpovědná za vykonání akce daného handleru. Tato metoda je volána, když uživatel vybere specifickou položku v menu. Dále může tato třída obsahovat metodu s anotací `@CanExecute`, která říká, jestli může být třída vykonána (viz ukázka kódu 5). Pokud vrací false, Eclipse zakáže příslušící element uživatelského rozhraní. Defaultně tato metoda vrací hodnotu true, což znamená, že pokud může být daný handler vždy vykonán, není potřeba metodu s touto anotací implementovat.

```
public class NewHandler {
   @Execute
   public void execute() {

   }

   @CanExecute
   public boolean canExecute() {
      return true;
   }
}
```

Výpis 5: Ukázka vygenerovaného handleru

V tuto chvíli je tedy vytvořen Command a Handler. Zbývá vytvořit danou položku v Menu, která bude tyto věci využívat. V naší vygenerované základní aplikaci již je jedno okno s dostupným hlavním menu, které obsahuje další 2 menu - File, Help. Tyto menu můžeme rozšiřovat, přidávat oddělovače, dále je větvit. Momentálně chceme přidat položku pro náš Command. Proto přidáme do menu položku Handled Menu Item, u níž nastavíme popisek a příslušný Command, který chceme, aby tato položka vykonávala (viz obrázek 15).

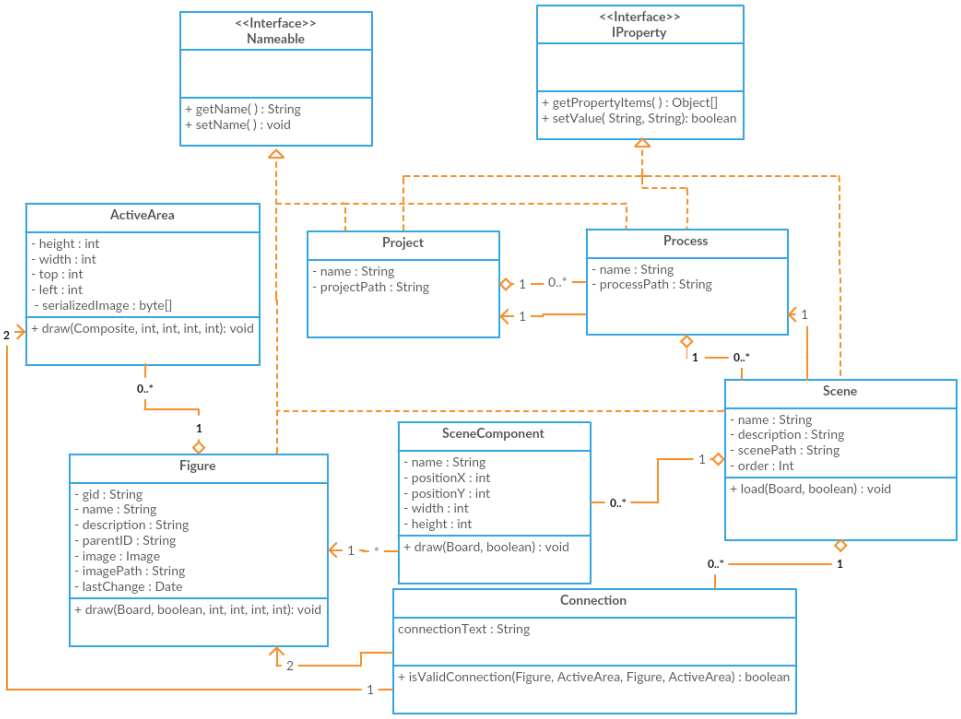


Obrázek 15: Vytvoření položky menu

4 Implementace

4.1 Rozšířený model struktury storyboardu

Pro realizaci modelu zobrazeného na obrázku 11 v kapitole 2.5 Struktura storyboardu, bylo potřeba tento model rozšířit o několik dalších tříd (viz obrázek 16).



Obrázek 16: Rozšířený model struktury tříd

Základní třídy Project, Process, Scene a Figure nyní implementují rozhraní Nameable, které objektům přidává vlastnost umožňující nastavení názvu objektu. Dalším rozhraním, které bylo do modelu přidáno, je rozhraní IProperty. To přidává objektům metody využívané v aplikaci v okně Properties a umožňuje zobrazit vlastnosti objektu a dále i případnou editaci těchto vlastností.

Třída Figure jako taková reprezentuje pouze jednotlivé artefakty, ze kterých se scéna může seskládat. Může sice obsahovat informace o rozměrech artefaktu, ovšem při přidání figure na scénu je nezbytné ukládat i umístění dané figure v této scéně. Z tohoto důvodu byla přidána třída SceneComponent mezi Figure a Scene, která tedy obsahuje figure, který reprezentuje, ale také její umístění na scéně.

Dále bylo potřeba přidávat do figure i aktivní oblasti, které by rozšiřovaly využití Storyboard editoru o možnost propojování jednotlivých figure na scéně. Z tohoto důvodu byla přidána třída ActiveArea, která reprezentuje jednotlivé aktivní oblasti v dané figure. Třída ActiveArea zajišťuje uložení informací týkajících se umístění a rozměrů dané aktivní oblasti, dále identifikátor

pro zajištění individuality aktivních oblastí, seznam identifikátorů figure, se kterými je možné vytvářet spojení, a seznam řetězců pro popis případného vytvořeného spojení.

Pro reprezentaci vytvořených spojení následně slouží třída Connection. Ta obsahuje tedy popis spojení, aktivní oblasti, které vytváří toto spojení, a také figure, kterým jsou tyto aktivní oblasti přiřazeny.

4.2 Rozdělení balíčků

- handlers - Balíček, který obsahuje třídy starající se o funkcionalitu položek hlavního menu.
- listOfImages - Balíček obsahující třídy sloužící pro získání dat potřebných pro vykreslení seznamu figures.
- model - Balíček tříd, které reprezentují strukturu storyboardu.
- modeling - Balíček obsahující třídy pro vykreslení scén a detekci kolizí.
- parts - Balíček, jehož třídy reprezentují jednotlivé části aplikace a využívají tříd ostatních balíčků pro zobrazení uživatelského rozhraní.
- preferences - Balíček tříd starajících se o veškerá nastavení aplikace.
- propertyStructure - Balíček obsahující třídy určené pro zobrazení okna Properties pro objekty.
- service - Balíček, který obsahuje třídy starající se o ukládání a načítání objektů do paměti a třídy starající se o export procesů.
- svg - Balíček tříd určených k získání obrázku z formátu SVG.
- treeStructure - Balíček, obsahující třídy sloužící ke správnému zobrazení a funkcionalitě stromových struktur v aplikaci.
- utils - Balíček pomocných tříd využívaných např. při práci s ukládáním a načítáním objektů.

4.3 Service - ukládání a načítání

Veškeré ukládání a načítání objektů do paměti (CRUD operace) je řešeno v balíčku service. Centralizováním ukládání a načítání je otevřena cesta k různému způsobu ukládání.

V aktuální verzi jsou veškerá data ukládány do XML souborů. Pro práci s XML se využívá JAXB (Java Architecture for XML Binding), která využívá JAXB anotací pro převádění java objektů do a z XML souborů. Zde se využívá takzvaný marshalling pro převádění objektů do XML souborů a unmarshalling pro převádění obsahu XML souboru do Java objektů.

Pro objekty, které se ukládají do nebo načítají z XML, je třeba přidat anotace. Mezi tyto anotace patří:

- `@XmlRootElement` - Definuje kořenový element pro vytvářené XML. Tato anotace se nachází před definicí třídy. Název kořenového elementu je odvozen z názvu třídy, případně lze nastavit u anotace atribut name pro jiný název).
- `@XmlType` - Určuje pořadí, v jakém jsou vlastnosti ukládány (např. `@XmlType(propOrder = \{ "id",`
- `@XmlElement` - Tato anotace se vkládá před vlastnosti objektu, které chceme uložit do XML jako elementy v kořenovém elementu.
- `@XmlAttribute` - Tato anotace má podobný význam jako `@XmlElement`, pouze je element uložen jako atribut kořenového elementu.
- `@XmlTransient` - Označuje vlastnosti objektu, které chceme při ukládání ignorovat, tzn. ty, které nechceme, aby byly uloženy v XML souboru.

Pokud chceme uložit uvnitř elementu například další kolekci jiných objektů, používají se ještě notace `@XmlElements` a `@XmlElementWrapper` (viz ukázka kódu 6).

```
@XmlElements({ @XmlElement(name = "activeArea", type = ActiveArea.class), })
   @XmlElementWrapper
   public List<ActiveArea> getActiveAreas() {
      return activeAreas;
   }
```

Výpis 6: Ukázka využití notací @XmlElements a @XmlElementWrapper

Jednotlivé metody service jsou zodpovědné za to, že načtou/uloží objekt celý včetně všech navázaných částí. Pro uložení objektu je volána metoda marshall z třídy FileUtils, která je v balíku utils (viz ukázka kódu 7). Jako parametry bere metoda třídu ukládaného objektu, daný objekt a soubor, do kterého se má uložit. V metodě se poté vytvoří vstupní bod k JAXB API, marshaller pro uložení objektu a následně se objekt uloží do daného souboru.

```
public static void marshall(Class clasz, Object object, File file) throws
    JAXBException {
      JAXBContext jaxbContext = JAXBContext.newInstance(clasz);
      Marshaller jaxbMarshaller = jaxbContext.createMarshaller();
      jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

      jaxbMarshaller.marshal(object, file);
   }
```

Výpis 7: Ukázka uložení objektu pomocí JAXB

Pro načtení objektu z XML je volána naopak metoda unmarshall, která bere jako parametry třídu načítaného objektu a soubor, ze kterého načítá objekt (viz ukázka kódu 8). Následně je

opět vytvořen vstupní bod k JAXB API, unmarshaller pro načtení objektu a nakonec je načten objekt z XML.

```
public static Object unmarshall(Class clasz, File file) throws JAXBException {
      JAXBContext jaxbContext = JAXBContext.newInstance(clasz);
      Unmarshaller jaxbUnMarshaller = jaxbContext.createUnmarshaller();

      Object returnObject = jaxbUnMarshaller.unmarshal(file);
      return returnObject;
   }
```

Výpis 8: Ukázka načtení objektu pomocí JAXB

4.3.1 Ukládání figure

Vytvořené figure se ukládají do složky podle nastavení figurePath v Preferences oknu (to je popsáno v kapitole 4.6.6). Výsledný soubor má název odpovídající uuid figure. Jeho obsahem je uuid, název, rozměry, rodičovský figure, svg kód figure, označení, zda se jedná o element na pozadí, datum a čas poslední změny figure a seznam aktivních oblastí (ty mají své id, rozměry, umístění ve figure, svg kód a seznamy uuid propojitelných figure a řetězců pro vytváření connections (viz ukázka kódu 9).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<figure uuid="box01">
    <activeAreas>
        <activeArea>
            <connectables>
                <connectable>Worker</connectable>
            </connectables>
            <connectionStrings>
                <string>creates</string>
            </connectionStrings>
            <height>50</height>
            <id>1</id>
            <left>38</left>
            <SVGCode>&lt;svg xmlns="http://www.w3.org/2000/svg" ...</SVGCode>
            <top>16</top>
            <width>50</width>
        </activeArea>
    </activeAreas>
    <backgroundElement>false</backgroundElement>
```

```
    <height>200</height>
    <lastChange>2017-03-07T10:19:14.659+01:00</lastChange>
    <name>Object</name>
    <parentID></parentID>
    <SVGCode>&lt;svg xmlns="http://www.w3.org/2000/svg" ...</SVGCode>
    <width>200</width>
</figure>
```

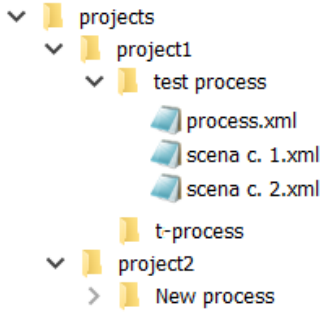
Výpis 9: Ukázka uloženého figure

4.3.2 Ukládání projektů, procesů a scén

Projekt jako takový je reprezentován složkou v souborovém systému. V jedné projektové složce se nachází složky, které reprezentují jednotlivé procesy.

Ve složce procesu je vytvořen soubor `process.xml`, který popisuje konkrétní proces, a dále soubory, které které reprezentují jednotlivé scény (viz obrázek 17).

V každém projektu může být uložen libovolný počet procesů a v každém procesu libovolný počet scén.



Obrázek 17: Projektová struktura

4.4 Práce s SVG

4.4.1 SVG

SVG je zkratkou pro Scalable Vector Graphics neboli škálovatelnou vektorovou grafiku. Jedná se o značkovací jazyk a formát souboru používaný k ukládání vektorové grafiky.[3]

Oproti běžným formátům obrázku jako je JPEG, PNG nebo GIF má SVG formát určité výhody:

1. SVG obrázky mohou být vytvářeny a editovány libovolným textovým editorem.
2. SVG obrázky jsou škálovatelné.

3. SVG obrázky mohou být zvětšeny či zmenšeny bez ztráty kvality obrázku.

4. SVG obrázky jsou přesně popsány pomocí XML značek.

SVG soubory mohou být vytvořeny libovolným textovým editorem, ovšem příhodnější je pro vytváření využívat některý program pro kreslení SVG obrázků, jako je například program Inkscape.

4.4.2 Instalace

V rámci projektu byl jedním z požadavků import obrázků ve formátu SVG. Existuje několik volně dostupných knihoven, které práci s SVG řeší. Mezi nejznámější patří asi Batik SVG Toolkit. O využití této knihovny jsme se v týmu pokoušeli dlouhou dobu, ovšem vzhledem ke komplexnosti a provázanosti jednotlivých modulů a chybě v gradle wuff projektu se nám knihovnu Batik nepodařilo zprovoznit. Nakonec jsme zvolili jinou knihovnu, která ovšem svou funkčností, kterou jsme potřebovali, byla dostačující. Touto knihovnou je SVG Salamander.

Verze, která je ve standardních maven repozitářích byla ale velmi stará, a proto jsme se rozhodli si knihovnu kompilovat svépomocí. Také jsme chtěli odstranit závislosti na Apache Ant, která v knihovně slouží pro přímé zpracování SVG.

Abychom toto mohli udělat, naimportovali jsme zdrojové kódy z oficiální SVN do vlastního GIT repozitáře a odstranili závislost.

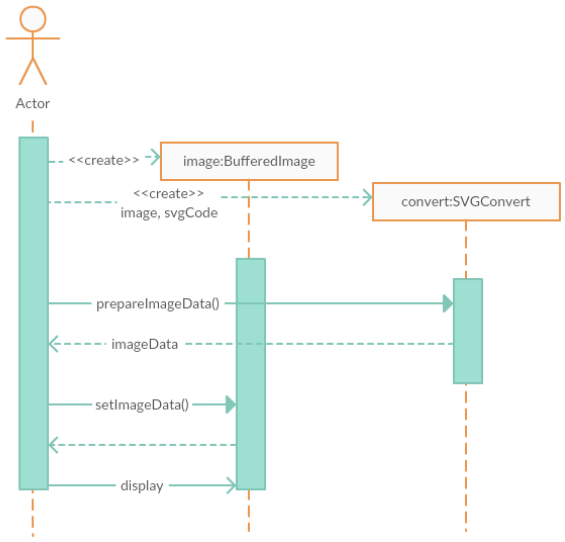
Ve vlastním projektu jsme poté nastavili tento repozitář jako submodul a vytvořili gradle task na jeho kompilaci (viz ukázka kódu 10). Pro jednoduchost jsme takto zkompilovanou knihovnu vložili přímo do projektu.

```
apply plugin: 'maven' //pro buildeni salamanderu

task buildsvgsalamender(type: Exec){
   commandLine 'cmd', '/c' ,'mvn package -f ../svg-salamander/svg-core/pom.xml'
}
```

Výpis 10: Vytvoření task pro SVG Salamander

4.4.3 Použití

Pro převod SVG kódu na zobrazitelný obrázek je potřeba vytvořit objekt typu BufferedImage s konkrétní šířkou a výškou. Následně je nutné vytvořit objekt typu SVGConverter a tomu vytvořený image předat společně s SVG kódem, který chceme převést. Zavoláním metody prepareImageData ze třídy SVGConverter se následně SVG kód, který byl předán v konstruktoru, konvertuje na objekt typu ImageData a tyto data vrátí. Následně lze tyto imageData nastavit jako data image a tento image poté zobrazit (viz obrázek 18).



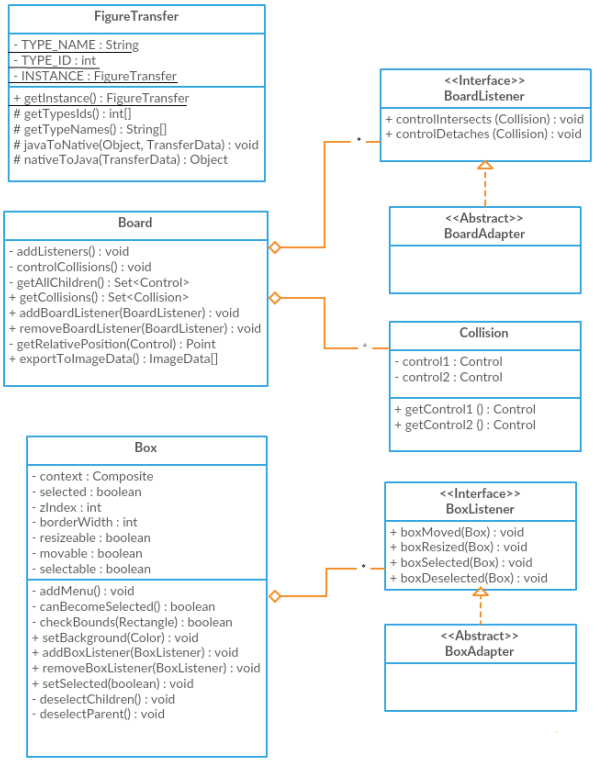
Obrázek 18: Sekvenční diagram převodu SVG obrázku

Při změně velikostí je potřeba tento postup opakovat. Je ideální zvolit hranici mezi pěkným obrázkem a opakovaným převodem.

Třídě SVGConvert je možné předat místo SVG kódu i objekt typu File, ve kterém je SVG kód uložen. Třída se poté sama postará o získání obsahu.

4.5 Modelování, vykreslování objektů

O modelování se stará balíček tříd modeling. Ten obsahuje potřebné třídy pro vykreslování a správu objektů na Canvasu (viz obrázek 19). Mezi nejdůležitější třídy tohoto balíku patří třída Board a třída Box. Na následujícím diagramu (obr. 19) jsou znázorněny všechny třídy obsažené v tomto balíčku.



Obrázek 19: Diagram tříd balíčku modeling

4.5.1 Board

Třída Board dědí ze třídy Canvas. Tato třída se stará o veškerou logiku týkající se vykreslování grafických objektů sloužících k reprezentaci scén procesu. Uvnitř této třídy dochází mimo jiné i k detekci kolizí mezi jednotlivými objekty a také obstarává MouseEvent pro rušení selekce objektů.

K detekci kolizí bylo potřeba použít PaintListener, který je volán vždy, když dojde k překreslení Boardu. K překreslení boardu dochází například vždy, když dojde k posunutí, přidání nebo změně velikosti vykresleného objektu na boardu. Na základě vyvolané události poté dojde k procházení vykreslovacího stromu a porovnávání, zda jsou některé dvojice objektů v kolizi. Výsledné kolize jsou poté uchovány v listu a porovnávány s předchozím listem kolizí, aby se detekovaly případné nově vzniklé nebo zaniklé kolize. Ty jsou následně propagovány všem posluchačům, kteří naslouchají BoardListeneru.

Dále je v této třídě naimplementována pomocná funkce getRelativePosition(), která zjišťuje relativní pozici vykresleného objektu vzhledem k boardu. V tomto případě se prochází všichni rodiče vykresleného objektu a připočítává se jejich X a Y pozice. Toto procházení skončí až v případě, kdy je rodič samotná instance třídy Board. Následně se výsledné pozice X a Y vrátí jako instance třídy Point.

Další funkcí v této třídě je exportToImageData(), která slouží pro vytvoření snímku daného boardu a jeho exportování do objektu reprezentujícího obrázek. Takto vytvořený objekt poté slouží k zobrazení náhledu boardu jednotlivých scén v procesu tak, aby měl uživatel celkový přehled a nemusel pro zobrazení proklikávat všechny scény.

4.5.2 Box

Druhou důležitou třídou balíčku modeling je třída Box, která úzce spolupracuje s třídou Board. Instance třídy Box totiž graficky reprezentují objekty vykreslené právě na boardu. Třída Box dědí z třídy Composite, takže přebírá všechnu potřebnou funkcionalitu pro správné vykreslení objektu.

Třída obsahuje i další objekt třídy Composite, který slouží k zobrazení obsahu. V tomto projektu tedy pro zobrazení obrázku ve formátu SVG. Samotná třída Box vykresluje pouze okraj, který uživateli ukazuje, zda byl daný objekt vybrán pomocí kliknutí myši. Důvodem pro toto vnoření dvou Composite objektů bylo právě to, aby docházelo ke správnému vykreslování a nedocházelo tak k chybám v překrývání a jiným artefaktům.

V této třídě jsou také zpracovány všechny potřebné akce pomocí posluchačů na jednotlivé události myši. Mezi tyto akce patří zvětšení nebo zmenšení objektu, posun objektu a také jeho selekce. Dále je pomocí pravého tlačítka myši vyvolána akce pro zobrazení menu objektu.

Dále třída obsahuje potřebnou implementaci menu pro ovládání již vykreslených prvků na boardu. Toto menu obstarává funkcionalitu v závislosti na objektu, kterému box patří. Pro figure poskytuje možnost upravení vlastností, která otevře dialogové okno pro úpravu vlastností figure, který je boxem reprezentován. Dále menu nabízí možnost smazání objektu, která vyvolá metodu dispose, která se postará, aby byl objekt správně odstraněn a dealokován. Menu ještě nabízí dále také možnost pro posunutí objektu do popředí a možnost pro posunutí objektu do pozadí. To se provádí změnou hodnoty v třídní proměnné zIndex, která určuje Z pozici objektu vzhledem k ostatním vykresleným objektům. Pro aktivní oblasti poskytuje menu možnost upravení popisku connection, pokud existuje.

Při řešení, jak jednotlivé objekty posouvat či měnit jejich velikost, byla použita funkcionalita třídy Tracker. Tato třída je obsažena přímo ve frameworku SWT a obstarává právě toto chování. Nejprve je potřeba odchytit událost vyvolanou MouseEventem a poté vytvořit novou instanci třídy Tracker. Poté se pomocí funkce setRectangles() přidal výchozí rozměr pro přesouvaný či zvětšovaný objekt. Po dokončení MouseEventu došlo poté ke změně cílového objektu, ať už byl zvětšen či jen přesunut (viz ukázka kódu 11).

```
Tracker tracker = new Tracker(getParent(), SWT.UP | SWT.LEFT | SWT.DOWN | SWT.
    RIGHT);
tracker.setStippled(false);
tracker.setRectangles(new Rectangle[] { getBounds() });
if (tracker.open()) {
```

```
    Rectangle after = tracker.getRectangles()[0];
    setBounds(after);
    boxMoved();
}
tracker.dispose();
```

Výpis 11: Využití třídy Tracker

Dále ještě třída obsahuje i metody pro zobrazení textu při najetí kurzorem myši nad daný objekt. V případě, že je objektem box reprezentující figure, zobrazí se název daného figure. V případě, že box reprezentuje aktivní oblast, tak pokud je tato aktivní oblast součástí nějaké connection, zobrazí se popisek této connection.

4.5.3 Collision

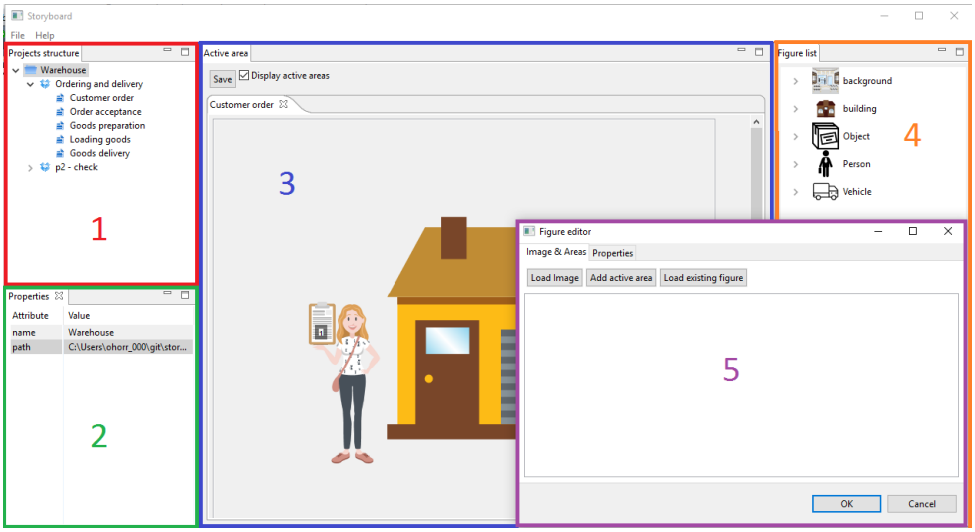
Třída Collision slouží k reprezentaci kolizí na boardu. Obsahuje dvě proměnné - collision1 a collision2. V těchto proměnných jsou uchovány objekty, které jsou spolu v kolizi. Dále tato třída obsahuje implementaci funkce equals, aby se zabránilo duplicitám. Tato funkce řeší jedinečnost dvojice, tzn. že pokud je objekt 1 v kolizi s objektem 2, tak je to stejné, jako kdyby byly tyto dva objekty prohozeny. Tím docílíme, že pokud se instance této třídy následně uloží do Setu, získáme kolekci, která nebude obsahovat duplicitní kolize.

4.5.4 FigureTransfer

Třída FigureTransfer slouží pro zajištění funkcionality Drag and Drop akce. Pro tuto funkcionalitu je potřeba dědit ze třídy ByteArrayTransfer a poté definovat nový datový typ pro Drag and Drop akci. Pro tuto definici slouží statická konstanta `TYPE_NAME`, která nabývá hodnoty "FigureTransfer". Dále bylo potřebné nastavit i další statickou proměnnou s názvem `TYPE_ID`, jejíž hodnota je vygenerována pomocí funkce registerType. Protože ByteArrayTransfer implementuje návrhový vzor Singleton, tak i ve třídě FigureTransfer musíme uchovávat pouze jedinou instanci této třídy. Ta je uložena ve statické konstantě instance.

Po definování všech třídních konstant bylo potřeba zajistit samotnou funkcionalitu pro Drag and Drop. K tomu slouží funkce javaToNative a nativeToJava. První z funkcí převádí objekt, ze kterého provádíme Drag, na binární datovou reprezentaci. Druhá z funkcí následně převádí binární datovou reprezentaci objektu zpět na objekt z Javy.

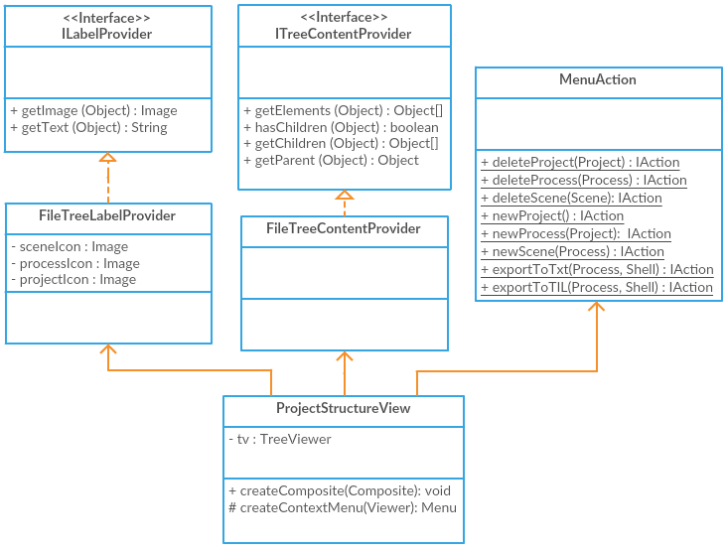
4.6 Popis částí aplikace



Obrázek 20: Ukázka aplikace a rozdělení na části

Aplikace by se dala rozdělit do pěti nejdůležitějších částí (viz obrázek 20). První část tvoří stromová struktura projektů, procesů a scén vytvářených v aplikaci a umožňuje přidávání nových projektů, procesů a scén. Druhá část zobrazuje přehled vlastností objektu vybraného ve stromové struktuře projektů a umožňuje jejich editaci. Třetí část tvoří nejdůležitější část aplikace a to je okno, ve kterém se vytváří jednotlivé scény popisující proces. Čtvrtá část slouží pro hierarchické zobrazení vytvořených figure, ze kterých lze sestavit scénu. Pátou část následně tvoří samotný editor pro vytváření nových a úpravu již existujících figure.

4.6.1 Popis části Project structure



Obrázek 21: Diagram tříd pro část Project structure

Hlavní třídou této části je třída ProjectStructureView sloužící pro zobrazení a funkcionalitu obsahu prvního dílu aplikace. Poskytuje možnost zobrazení struktur projektů a umožňuje také vytváření nových a mazání stávajících projektů, procesů a scén. Aby bylo toto zobrazení přehledné a intuitivní, bylo využito třídy TreeViewer, která jak již název napovídá slouží pro vizualizaci určité struktury ve formě stromu (viz obr 22). Jelikož jsou v podobné formě řešeny struktury projektů, adresářů, atd. i v jiných aplikacích, nebylo zapotřebí moc přemýšlet nad jiným způsobem zobrazení, který by mohl vést jedině k disorientaci uživatele.

Pro vytvoření treeViewer je potřeba poskytnout dva providery. Prvním je provider poskytující zobrazovaná data, druhým je posléze provider pro zobrazení dat v námi požadovaném tvaru.

Pro získání zobrazovaných dat byla vytvořena třída FileTreeContentProvider, která implementuje rozhraní ITreeContentProvider. Toto rozhraní přidává metody potřebné pro poskytnutí obsahu pro námi zobrazovaný strom. Mezi tyto metody se řadí převážně metoda getElements, která načte objekty zobrazované ve stromu. V našem případě prochází složku s projekty, které načte do paměti a následně vrátí seznam těchto projektů. Mezi další důležité metody také patří metoda hasChildren, která oznamuje, zda má objekt ještě další zanořené objekty pro zobrazení, a metoda getChildren, která vrací tyto zanořené objekty pro určitý rodičovský objekt. V našem případě tedy v momentě, kdy je jako rodič poslán projekt, vrátí tato metoda seznam procesů tohoto projektu. V případě, kdy je rodičem proces, vrací metoda seznam scén daného procesu.

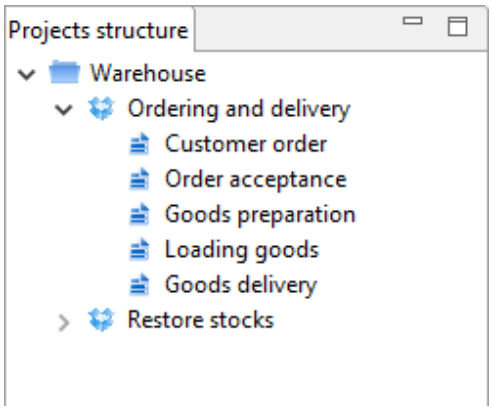
Tím by byl vytvořen provider poskytující načtená data ve specifickém formátu. Dále bylo zapotřebí vytvořit třídu, která určí, jak budou zobrazovaná data vypadat. Za tímto účelem je vytvořena třída FileTreeLabelProvider, která musí implementovat rozhraní ILabelProvider. Zde

stojí za zmínku převážně metody getText vracející text reprezentující objekt a getImage, která pro daný objekt vrací obrázek ikony.

Ve třídě ProjectStructureView jsou poté tyto providery přiděleny k vytvořené instanci třídy TreeViewer (viz ukázka kódu 12).

```
tv = new TreeViewer(parent, SWT.H_SCROLL | SWT.V_SCROLL | SWT.SINGLE);
      tv.getTree().setLayoutData(new GridData(GridData.FILL_BOTH));
      tv.setContentProvider(new FileTreeContentProvider());
      tv.setLabelProvider(new FileTreeLabelProvider());
```

Výpis 12: Nastavení providerů pro TreeViewer



Obrázek 22: Zobrazení projektové struktury v aplikaci

V tomto okamžiku byl sice vytvořen strom zobrazující projekty, procesy a scény, ovšem bylo nezbytné doplnit i nějakou funkcionalitu pro tento strom. Jednou z funkcionalit bylo poskytnutí možností vytváření a odebírání projektů, procesů a scén. Tyto možnosti byly nejprve vloženy do nabídky hlavního menu, ovšem z této varianty rychle sešlo kvůli nepřehlednosti a horší manipulaci. Z tohoto důvodu byly tyto možnosti přesunuty z hlavního menu do kontextového menu u vytvořeného stromu. Pro přidání kontextového menu se využila třída MenuManager, která slouží pro vytvoření menu pro objekty. Pro zobrazení menu je nutné přiřadit k instanci této třídy MenuListener, který pomocí metody menuAboutToShow(IMenuManager mgr) určuje, jak by mělo menu vypadat. V našem případě je na základě vybraného objektu ve stromu nastaveno příslušné kontextové menu umožňující dané akce pro příslušný objekt. Pro tyto akce kontextového menu byla vytvořena třída MenuAction, která poskytuje implementaci jednotlivých akcí kontextových menu napříč aplikací. Menu našeho treeVieweru nabízí možnosti přidání a odebrání projektů, procesů a scén a také možnost exportu vybraného procesu do textového nebo TIL Script formátu.

4.6.1.1 Export vytvořených storyboardů

Jedním z požadavků byl také export storyboardů do textového formátu a do formátu TIL Script.

Pro ukládání textového formátu slouží třída TextExportService. Do jednoho textového souboru se uloží popis vždy celého vybraného procesu. Ukládá se název procesu a následně pro každou scénu vždy název scény, popis scény a poté seznam vytvořených connections (viz obrázek 23).

```
Process: Ordering and delivery

Scene: Customer order
Description: Customer wants to buy something, so he creates an order.

Customer creates Order

Scene: Order acceptance
Description: Order is accepted, processed and sent to warehouse.

manager accepts Order
manager creates delivery plan
```

Obrázek 23: Ukázka exportu do textového formátu

Druhým způsobem exportu byl export do formátu TIL Script. TIL (Transparent Intensional Logic) je originální logicko-sémantický systém, jehož základy položil český logik Pavel Tichý a dále ji i rozpracoval do mimořádné hloubky a šířky. TIL nabízí procedurální sémantiku, díky čemuž se vyhýbá mnohým úskalím (např.) intenzionálních sémantik budovaných na množinové bázi. Vzhledem na nesmírnou bohatost a komplikovanost sémantických jevů v přirozených jazycích je TIL systém, který nadále rozvíjí a rozšiřuje více logiků a filozofů (nejen) z České republiky. TIL je užitečný v oblasti sémantiky přirozeného jazyka, dále v oblasti aplikace v počítačových vědách, např. datový model HIT, nebo je také užitečný např. pro vysvětlení některých filozofických pojmů (vlastnosti, koncepty, konceptuální systémy).[8]

V projektu storyboard je tento formát využit k exportu connections u scén v procesu. Každý connection je uzavřený v hranatých závorkách, jednotlivé elementy jsou odděleny mezerou a začínají apostrofem. První element tvoří text daného connection, druhý a třetí element následně tvoří názvy figure, které tento connection tvoří (viz obrázek 24.

```
['Creates 'Customer 'Order]

['Accepts 'Manager 'Order]
['Creates 'Manager 'Delivery_plan]
```

Obrázek 24: Ukázka exportu do formátu TIL Script

4.6.1.2 Zasílání událostí

Nakonec bylo nezbytné ještě zařídit, aby se selekce objektů v našem stromu nějakým způsobem distribuovala i do jiných View, které na základě selekce prováděly svou vlastní činnost. Takto distribuované události se ovšem netýkaly jen této části aplikace, nýbrž i celkově komunikaci mezi view v aplikaci.

Jelikož není možné se dostat přímo k instancím jednotlivých view, nebylo možné poslat data přímo pomocí nějaké metody do dalšího view. Jedině, pokud by metoda byla statická, což ovšem není zrovna ideální řešení. Proto jsem se snažil vyzkoušet řešení pomocí selectionProvideru a selectionListeneru, které se mi ale ani po delší době nedařilo zprovoznit, jelikož tento způsob využíval Eclipse 3, zatímco projekt Storyboard je vytvořen jako e4 projekt.

Následně bylo zasílání informací mezi view vyřešeno pomocí OSGi služeb umožňující registrování, zasílání a zpracování událostí. Pro zaslání události bylo nejprve nutné získat BundleContext dané třídy. Poté bylo třeba získat instanci služby EventAdmin, pomocí které se bude událost distribuovat. Dále se musela vytvořit mapa vlastností, které se v události zasílají. Do této mapy se uloží k danému textovému klíči zasílaná data. Následně je vytvořena událost, které se tato mapa vlastností přiřadí a která je nakonec distribuována pomocí eventAdmina (viz ukázka kódu 13).

```
BundleContext ctx = FrameworkUtil.getBundle(ProjectStructureView.class).
    getBundleContext();
ServiceReference<EventAdmin> ref = ctx.getServiceReference(EventAdmin.class);
EventAdmin eventAdmin = ctx.getService(ref);

Map<String, Object> properties = new HashMap<String, Object>();
List<Nameable> data = new ArrayList<Nameable>();
IStructuredSelection selection = (IStructuredSelection) event.getSelection();
Object obj = selection.getFirstElement();
if (obj instanceof Nameable) {
   data.add((Nameable) obj);
}
properties.put("DATA", data);

Event ev = new Event("parts/ProjectStructureView", properties);
eventAdmin.sendEvent(ev);
```

Výpis 13: Zasílání eventů v aplikaci

V jiném view přijímajícím tyto distribuovaná data se musí poté opět získat BundleContext dané třídy, ale oproti odesílateli je v přijímacím view potřeba vytvořit EventHandler pro zpracování přijímaných dat. Dále je vytvořen ještě slovník vlastností, ve kterém je popsáno, které typy zasílaných událostí budou odchytávány. Nakonec je vytvořený eventHandler spolu se slovníkem zaregistrován v BundleContextu (viz ukázka kódu 14).

```
BundleContext ctx = FrameworkUtil.getBundle(ActiveAreaView.class).
    getBundleContext();
EventHandler handler = new EventHandler() {
   @Override
```

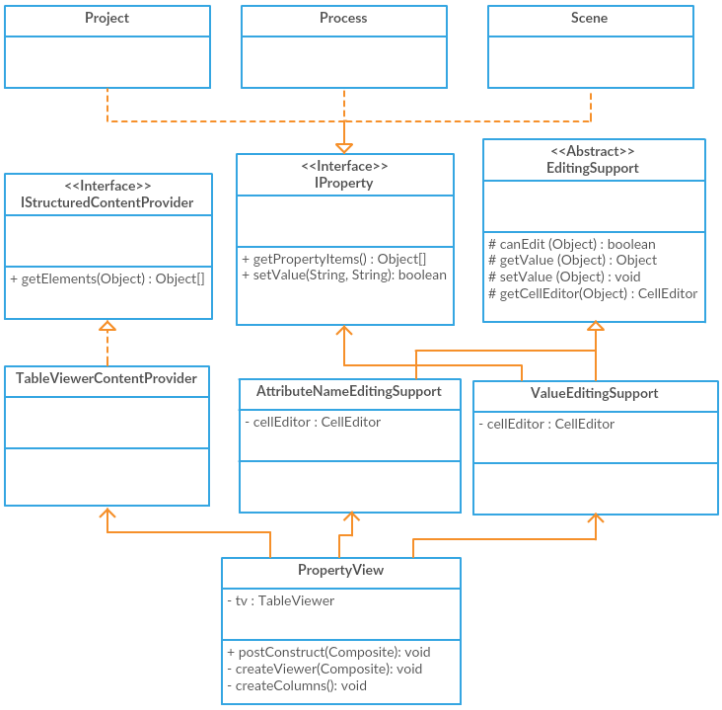
```
   public void handleEvent(final Event event) {
      if (parent.getDisplay().getThread() == Thread.currentThread()) {
         List<Nameable> selectedFiles = (ArrayList<Nameable>) event.getProperty(
             "DATA");
         // ... dalsi zpracovani ziskanych dat ...
      }
   }
};

Dictionary<String, String> properties = new Hashtable<String, String>();
properties.put(EventConstants.EVENT_TOPIC, "parts/ProjectStructureView");
ctx.registerService(EventHandler.class, handler, properties);
```

Výpis 14: Odchytávání eventů v aplikaci

4.6.2 Popis části Properties

Část Properties je druhou částí aplikace a slouží pro zobrazení a editaci vlastností projektů, procesů a scén. Vlastnosti jsou vždy zobrazeny pro objekt, který je aktuálně zvolen ve stromu projektů.

První snaha byla o vytvoření okna vlastností pomocí tříd implementujících rozhraní IAdaptable a IPropertySource. Bohužel se mi toto řešení v naší aplikaci ale nepodařilo ani po delší době rozchodit. Z toho důvodu jsem se rozhodl vytvořit a přizpůsobit si vlastní okno vlastností (diagram tříd viz obrázek 25).



Obrázek 25: Diagram tříd pro část Properties

Prvním krokem k vytvoření vlastního okna vlastností bylo vytvoření rozhraní obsahujícího metodu pro načtení položek okna vlastností a metodu pro ukládání změněných hodnot. Tímto rozhraním je IProperty a implementují ho třídy, pro které chceme mít možnost upravovat vlastnosti v okně Properties. V mém případě jsem chtěl, aby byly zobrazovány vlastnosti objektu, který byl momentálně zvolen ve stromu projektů. Z toho důvodu musí rozhraní IProperty implementovat třídy Project, Process a Scene. V těchto třídách jsou poté naimplementovány metody getPropertyItems a setValue. V metodě getPropertyItems se vytváří pole dvojic řetězců, kde první řetězec je název vlastnosti a druhým poté hodnota vlastnosti (viz ukázka kódu 15).

```
@Override
public Object[] getPropertyItems() {
   ArrayList<String[]> properties = new ArrayList<String[]>();
   String[] name = {"name", this.getName()};
   String[] path = {"path", this.getProjectPath()};
   properties.add(name);
   properties.add(path);
   return properties.toArray();
}
```

Výpis 15: Ukázka implementace metody getPropertyItems()

V metodě setValue se následně ukládají hodnoty. Parametry této funkce tvoří řetězec attribute určující, která vlastnost se má měnit, a řetězec data, který obsahuje novou hodnotu dat. V metodě se následně na základě názvu vlastnosti změní hodnota této vlastnosti (viz ukázka kódu 16).

```
@Override
public boolean setValue(String attribute, String data) {
   switch(attribute) {
   case "name": {
      if(projectPath != "") {
         File project = new File(projectPath);
         String parentDir = project.getParentFile().getAbsolutePath();
         boolean success = project.renameTo(new File(parentDir + "/" + data));

         if(success){
            this.setProjectPath(parentDir+"/"+data);
            this.setName(data);
            log.info("Project name changed.");
            return true;
         }
         else{
            log.info("Project name couldnt be changed.");
         }
      }
   }
   }
   return false;
}
```

Výpis 16: Ukázka implementace metody setValue(...)

Druhým krokem bylo vytvořit třídu PropertyView, která se stará o vykreslení okna vlastností. V této třídě se vytváří instance třídy TableViewer, v níž budou vlastnosti zobrazeny. Pro tuto instanci se poté musely vytvořit jednotlivé sloupce pro zobrazení vlastností, tedy sloupec pro názvy vlastností a sloupec pro hodnoty. Každému tomuto sloupci bylo potřeba nastavit LabelProvider, který se stará o vypisování popisků, a také nastavit EditingSupport pro zpracování změn hodnot v tabulce.

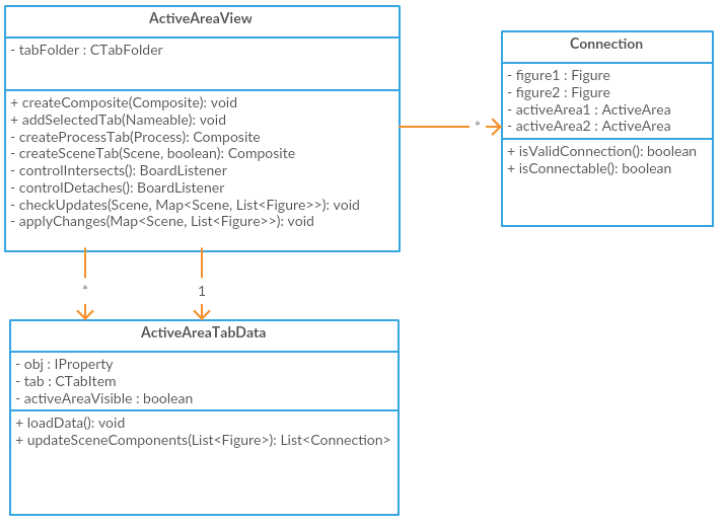
Pro manipulaci s daty v tabulce byly vytvořeny dvě třídy rozšiřující třídu EditingSupport. První třídou je AttributeNameEditingSupport, která se stará o sloupec s názvy vlastností. Tento sloupec nepotřebujeme editovat a proto vrací metoda canEdit v této třídě hodnotu false. Dále

v tomto sloupci chceme zobrazit názvy vlastností, takže metoda getValue vrací první z dvojice řetězců, které jsme vytvářeli ve třídách implementující rozhraní IProperty.

Druhou třídou je dále ValueEditingSupport, která se stará o sloupec s danými hodnotami vlastností. Zde je již editování povoleno a metoda getValue vrací hodnotu vlastnosti. Dále musí mít třída implementovanou i metodu setValue, která se stará o ukládání měněných hodnot. Uvnitř této metody se zavolá na objekt, pro který měníme vlastnosti, metoda setValue s danou vlastností a novou hodnotou.

Nakonec zbývalo k instanci třídy TableViewer připojit ještě ContentProvider, který získá pro vybraný objekt seznam zobrazitelných vlastností. Za tímto účelem byla vytvořena třída TableViewerContentProvider implementující rozhraní IStructuredContentProvider. Zde byla naimplementována metoda getElements, která pro daný objekt uložený v tableViewer získala seznam vlastností.

4.6.3 Popis části Active area



Obrázek 26: Diagram tříd pro část Active area

Jedná se o stěžejní okno celé aplikace, ve které se skládají a editují jednotlivé scény. O zobrazení a funkcionalitu se stará třída ActiveAreaView.

V tomto okně se zachytávají události zaslané ze stromové struktury projektů o vybraném objektu, pro který je následně vytvořena záložka a ta je zobrazena. Okno umožňuje zobrazit procesy a scény. V případě zvolení scény ve stromu je vytvořena záložka, která obsahuje instanci třídy Board a na které je zvolená scéna vykreslená. V případě, že je zvolený celý proces, vytvoří se záložka obsahující náhled na scény daného procesu a také záložky pro všechny scény v tomto procesu.

Pro zajištění, že je každá scéna načtena jen jednou, byla vytvořena třída ActiveAreaTabData, která uchovává informace o dané záložce a scéně, případně procesu, který k dané záložce patří.

Třída ActiveAreaView poté uchovává kolekci instancí této třídy, které reprezentují načtené scény. Při vytváření se pak prochází tato kolekce a pokud je již daný objekt načten, nenačítá se znovu, pouze se přepne záložka na tu, která obsahuje danou vykreslenou scénu.

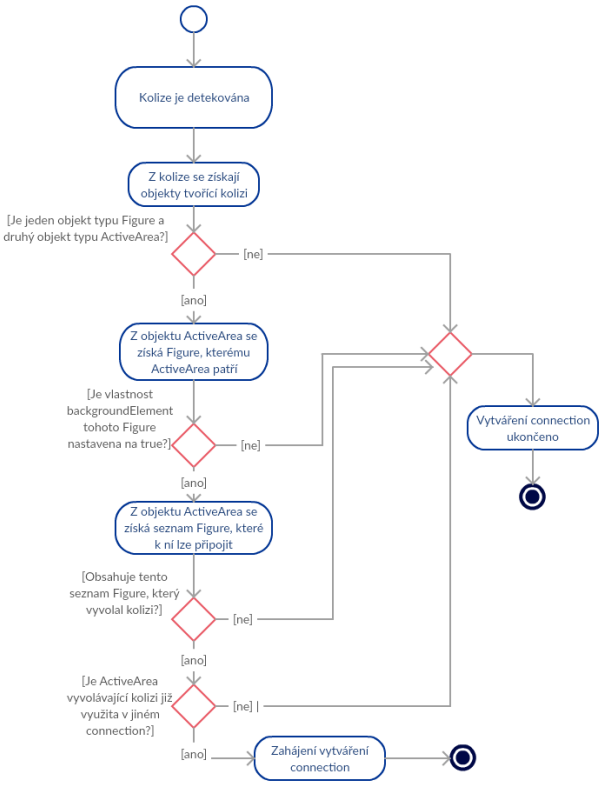
Při vytváření zobrazení kreslící plochy pro scénu se na board zaregistruje DropListener, který reaguje na přetažené figure z Figure listu. Také se na vznikající board registrují listenery na práci se vzniklými kolizemi - controlIntersect pro vytváření connection a controlsDetaches pro rušení connection.

Scéna také obsahuje tlačítko Save, které uloží právě aktivní scénu.

4.6.3.1 Popis vytváření connection

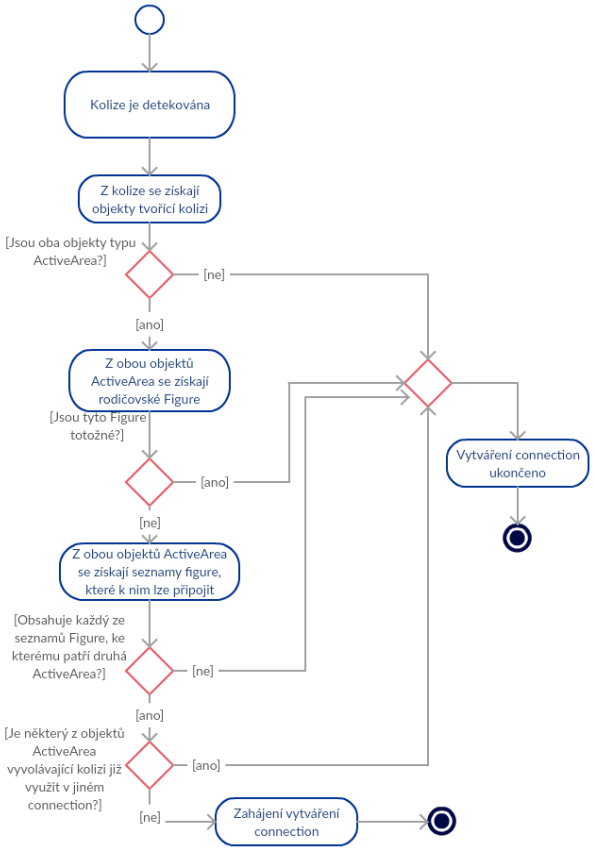
Registrováním listeneru controlIntersect na board se docílilo toho, že při vzniku kolizí na boardu se zavolá metoda `controlIntersects(Collision collision)` definovaná v tomto listeneru. V této metodě se nejprve získají objekty tvořící kolizi. Následně může nový connection vzniknout ve dvou případech.

V prvním případě se kontroluje, zda jeden objekt tvořící kolizi je instancí třídy ActiveArea a druhý je instancí třídy Figure. Pokud tomu tak je, tak se u Figure objektu kontroluje, zda má nastavenou vlastnost backgroundElement. Ta přidává objektům tu vlastnost, že daný figure nemusí být propojen s jiným figure jen pomocí aktivních oblastí, ale je možné propojit aktivní oblast tohoto figure přímo s jiným figure. Pokud tedy tuto vlastnost nastavenou má, pokračuje se do dalšího kroku. V tomto kroku se z aktivní oblasti tvořící kolizi získá seznam figure, které je možné k této aktivní oblasti propojit. Pokud figure připojovaný k této aktivní oblasti je v tomto seznamu figure, zobrazí se dialog pro vytvoření nového connection. Pokud není v tomto seznamu, ukončí se vytváření connection (grafické znázornění viz obrázek 27).



Obrázek 27: Diagram aktivit pro první případ

Ve druhém případě pro vytvoření connection se kontroluje, zda jsou oba objekty tvořící kolizi instancí třídy ActiveArea. Následně se kontroluje, zda figure, ke kterým patří tyto aktivní oblasti, nejsou totožné. Dále se z obou aktivních oblastí získají seznamy figure, které k nim lze připojit. V každém seznamu se poté kontroluje, zda obsahuje figure, který patří k druhé aktivní oblasti. Pokud některý ze seznamů aktivní oblasti neobsahuje figure, ke kterému se snaží připojit, ukončí se vytváření connection. V opačném případě se ještě kontroluje, zda již existující connection neobsahují některou z těchto aktivních oblastí. Pokud aktivní oblasti nejsou ještě využity, zobrazí se dialog pro vytvoření nového connection (grafické znázornění viz obrázek 28).



Obrázek 28: Diagram aktivit pro druhý případ

4.6.3.2 Popis rušení connection

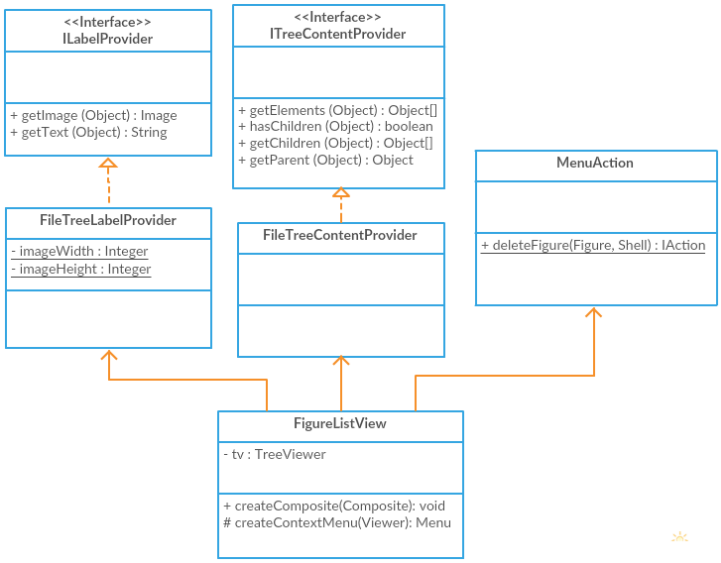
Rušení connection je řešeno přidáním listeneru controlsDetaches. Zde se při zániku kolize vyvolá metoda `controlsDetaches(Collision collision)` implementována v listeneru. Uvnitř této metody se podobně jako při vytváření connection kontroluje, zda je jeden objekt instance Figure a druhý ActiveArea, případně jestli jsou oba objekty instance ActiveArea. Po splnění některé z těchto podmínek se vytvoří nový objekt Connection, který se zkontroluje vůči listu existujících spojení. Pokud spojení existuje, zruší se daný connection a zobrazí se informační dialog o zrušení spojení.

4.6.3.3 Aktualizace figure

Při načítání scén se provádí i kontrola figures, které tvoří scénu. Je zkoumána doba poslední změny figure v porovnání s časem, kdy byl naposledy změněn daný figure ve figure listu. Pokud se najdou ve scéně figure s jiným datem poslední změny, zobrazí se uživateli dialog pro aktualizaci figure na scéně. Uživatel má poté na výběr, které figure chce aktualizovat. Následně jsou zvolené figure v dané scéně aktualizovány a je na scéně zavolána metoda updateConnectionList,

která kontroluje platnost existujících connections. Pro každý connection je poté zjišťováno, zda obsahuje aktualizovaný figure. Pokud obsahuje, kontroluje se nejprve existence aktivních oblastí tvořících connection. Následně se kontroluje, zda může být mezi danými aktivními oblastmi vytvořený connection a poté, zda jsou aktivní oblasti stále v kolizi. Pokud kolize mezi aktivními oblastmi není, zavolá se metoda fixConnection(Connection connection), která se pokouší o napravení connection pomocí posunutí figure na scéně. Pokud má pouze jedna aktivní oblast daného figure vytvořený connection, je tento figure posunut tak, aby docházelo mezi aktivními oblastmi ke kolizi a connection byla zachována. Nakonec jsou všechny connection, které nebylo možné zachovat, odebrány ze seznamu connections dané scény.

4.6.4 Popis části Figure list



Obrázek 29: Diagram tříd pro část Figure list

Tato část aplikace je určená pro zobrazení figure, ze kterých lze sestavit scénu. O zobrazení seznamu figure se stará třída FigureListView. Pro přehledné zobrazení figure bylo využito třídy TreeViewer podobně, jako tomu bylo u zobrazení struktury projektu. Pro tento TreeViewer bylo opět potřeba vytvořit contentProvider pro poskytnutí obsahu stromu a labelProvider pro poskytnutí popisků pro tyto objekty. Pro poskytnutí obsahu byla vytvořena třída FigureTreeContentProvider implementující rozhraní ITreeContentProvider. Zde jsou stěžejní metody getElements a getChildren.

Jelikož se k seznamu figures přistupuje nejen v této třídě, ale i v jiných částech aplikace, byla vytvořena i třída FigureLibrary, která načítá a uchovává kolekci všech figure. Díky tomu nemusí být při získání seznamu figure pokaždé znovu načítány všechny figures. Namísto toho se pouze z této třídy vytáhne již načtená kolekce. Třída FigureLibrary obsahuje i metody vracející určitou podkolekci všech figure, jako např. všechny potomky určitého figure.

Zajištění hierarchie mezi jednotlivými figures bylo zajištěno přidáním vlastnosti parent u figure. Ta obsahuje řetězec reprezentující uuid rodičovského figure a na základě této vlastnosti se vytváří i stromová struktura zobrazovaného seznamu figures.

Metoda getElements ve třídě FigureTreeContentProvider poté získává kolekci figures ze třídy FigureLibrary a z nich vybere pouze kořenové figure, tedy ty, které nemají žádný další rodičovský figure. Metoda getChildren následně pro určitý element načte ze třídy FigureLibrary pouze potomky tohoto figure.

Pro poskytnutí popisků pro objekty v instanci TreeViewer je dále vytvořena třída FigureTreeLabelProvider implementující rozhraní ILabelProvider. Zde jsou implementovány metody getText vracející název figure a getImage, ve které se načte SVG kód daného figure, ze kterého je vytvořena ikona pro daný figure.

Po nastavení providerů pro TreeViewer zobrazující hierarchii figure zbývalo ještě přidat funkcionalitu, kterou by se položky seznamu vkládaly na plátno v části Active area. Pro vkládání byla využita metoda drag-and-drop a proto musel být pro TreeViewer vytvořen DragSource a k tomu vytvořen listener pro tuto událost (viz ukázka kódu 17).

```
DragSource ds = new DragSource(tv.getControl(), DND.DROP_MOVE);
ds.setTransfer(new Transfer[] { FigureTransfer.getInstance() });
ds.addDragListener(new DragSourceAdapter() {
   @Override
   public void dragSetData(DragSourceEvent event) {
      IStructuredSelection selection = (IStructuredSelection)tv.getSelection();
      super.dragSetData(event);
      event.data = (Figure)selection.getFirstElement();
   }
});
```

Výpis 17: Vytváření DragSource

Na board musel být následně vytvořen DropTarget a k němu listener pro získání daného figure a vytvoření objektu SceneComponent, který reprezentuje figure umístěný na scéně (viz ukázka kódu 18).

```
DropTarget dropTarget = new DropTarget(board, SWT.NONE);
dropTarget.setTransfer(new Transfer[] { FigureTransfer.getInstance() });
dropTarget.addDropListener(new DropTargetAdapter() {
   @Override
   public void drop(DropTargetEvent event) {
      super.drop(event);

      if (event.data != null && event.data instanceof Figure) {
         Figure figure = (Figure) event.data;
```

```
        if (figure.getSerializedImage() != null) {
           Point point = board.toControl(event.x, event.y);
           SceneComponent sceneComponent;
           if(figure.getWidth() > 0 && figure.getHeight() > 0)
              sceneComponent = new SceneComponent(figure, point.x, point.y,
                  figure.getWidth(), figure.getHeight());
           else
              sceneComponent = new SceneComponent(figure, point.x, point.y,
                  200, 200);
           sceneComponent.draw(board, btnActiveAreaShow.getSelection());
                ((Scene)activeAreaTabData.getObject()).addSceneComponent(
                   sceneComponent);
        }
     }
  }
});
```

Výpis 18: Vytváření DropTarget

4.6.5 Popis části Figure editor

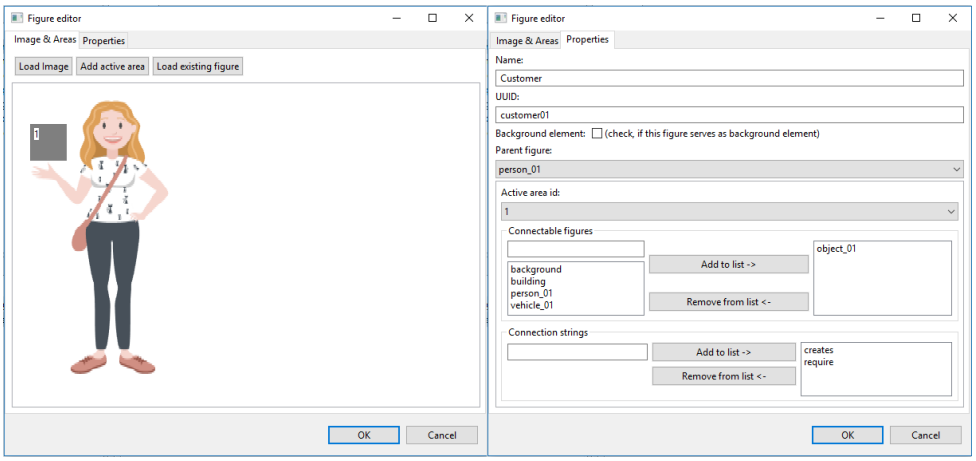
Figure editor je část aplikace umožňující vytváření nových a případnou editaci již existujících figure. Struktura editoru je nadefinována ve třídě EditPartDialog.

V okně editoru jsou dvě záložky. První je záložka Image & Areas a slouží pro vytvoření obrázku figure a aktivních oblastí. Druhou záložkou je poté záložka Properties umožňující nastavení dalších atributů (viz obrázek 30).

Vytvoření nové figure začíná v záložce Image & Areas načtením SVG souboru prostřednictvím tlačítka Load Image. Případně pro úpravu existujícího figure slouží pro načtení tlačítko Load Existing. Po načtení je obrázek vykreslen na plátno. Stisknutím tlačítka Add active area se vytvoří nová aktivní oblast, kterou lze posunout na libovolné místo. Dále lze měnit i rozměry načteného obrázku i jednotlivých aktivních oblastí. Následně se v záložce Properties definuje uuid, jméno, zda má mít figure vlastnost background element, tedy jestli mohou být aktivní oblasti této figure propojovány přímo s jinými figure a ne jen s aktivními oblastmi. Poté se definuje případný rodič pro daný figure. Nakonec se mohou nastavit ještě vlastnosti pro jednotlivé aktivní oblasti. U nich se nastavuje, které figure lze k dané aktivní oblasti připojit a také nabízené popisky při vytváření connection.

Při potvrzení vytváření figure se následně vytvoří nový figure nebo upraví existující. Dále se zkontrolují pozice a rozměry aktivních oblastí. Pokud se vyskytují někde mimo figure, jsou tyto aktivní oblasti posunuty, případně jsou upraveny jejich rozměry tak, aby se na daný figure

vešly. Vytvořený figure se následně uloží do místa předdefinovaného v nastavení aplikace (viz Preferences)



Obrázek 30: Ukázka vzhledu editoru

4.6.6 Popis Preferences

Část aplikace starající se o nastavení aplikace je soustředěna v balíčku preferences. Třída PreferencesView zobrazující okno pro editaci a uložení nastavení je poté v balíku parts. PreferencesView implementuje rozhraní IWorkbenchPreferencePage a dědí ze třídy FieldEditorPreferencePage. Díky těmto třídám je práce v ui nastavení jednoduchá a uživatelsky příjemná (vychází z eclipse.ui).

Eclipse framework ukládá změněná nastavení do souboru `workspace\.metadata\.plugins\org.eclipse.core.runtime\.settings\cz.vsb.storyboard.prefs`

Nastavení momentálně umožňuje pouze změnit cestu k složce s figure. Ovšem struktura je řešena tak, aby bylo jednoduché přidat v pozdějších verzích i další nastavení.

Inicializace nastavovaných hodnot se provádí ve třídě PreferenceInitializer. Ten rozšiřuje třídu AbstractPreferenceInitializer a implementuje metodu initializeDefaultPreferences. V ní lze nadefinovat výchozí hodnoty pro nastavení, ke kterým se případně dá vrátit pomocí tlačítka Restore Defaults v okně Preferences (viz ukázka kódu 19).

```
public class PreferenceInitializer extends AbstractPreferenceInitializer {
   @Override
   public void initializeDefaultPreferences() {
      IPreferenceStore store = Activator.getDefault().getPreferenceStore();
      store.setDefault(PreferencesConstant.FIGURE_PATH, "/sample-data/figure-
          library/");
   }
}
```

Výpis 19: Nastavení defaultních hodnot

Pro přístup k nastavením je využita pomocná třída Prefs, která obaluje získávání nastavení do formy, se kterou se v aplikaci dále pracuje.

5 Závěr

Informace obsažené v textu této práce měly přiblížit čtenáři jistou alternativní metodu pro modelování podnikových procesů a to metodu Storyboard, jejíž popisu se věnovala první kapitola práce. Další kapitoly poté měly za cíl seznámit čtenáře s již konkrétním návrhem a implementací editoru pro vytváření storyboardů.

Storyboard Editor byl vyvíjen v průběhu dvou let. Nejprve v rámci semestrálního projektu, na který následně navazovala tato diplomová práce. Hlavním přínosem již v rámci semestrálního projektu bylo naučit se spolupracovat v týmu, jelikož práce na tomto projektu v prvním roce byla prací tříčlenné skupiny. Díky tomu jsem si vyzkoušel, jaké to je pracovat v týmu a také jsem se díky tomu seznámil s verzovacím systémem Git, díky kterému byla správa kódů jednoduchá. Další přínos byl v naučení se nové technologie. Příkladem může být platforma Eclipse RCP nebo framework SWT.

Ovšem s frameworkem SWT se vyskytly i problémy. Jelikož SWT využívají nativní knihovny, chová se aplikace trochu jinak na každém systému. Například na systému OS X docházelo při přesouvání objektů na scéně ke správnému vykreslování okrajů, zatímco na Windows se tyto okraje vůbec nezobrazují. Dalším problémem bylo volání základního dialogového okna, kdy na operačním systému Windows vše fungovalo v pořádku, ale v OS X celá aplikace zamrzla a okno se zobrazilo až po 30 sekundách a bez všech prvků. Nakonec byl editor vyvíjen převážně na systému Windows, kde byl také testován. Na jiných operačních systémech by mohlo docházet k již zmíněným problémům.

V budoucích rozšířeních bych se zaměřil převážně na již zmíněné problémy týkající se jednotlivých operačních systémů a dále také na rozšíření možností exportu storyboardů, které by se daly využít např. i v jiných aplikacích. Dále by se mohl vytvořit i editor pro vytváření SVG obrázků, ze kterých se vytváří jednotlivé figure.

Horáček Ondřej

Literatura

[1] ManagementMania.com, *Podnikový proces (Business process)* [online]. Dostupné z: `https://managementmania.com/cs/business-process-podnikovy-proces`, [cit. 19.04.2017]

[2] BrightHubPM.com, *Some Examples of Storyboards in Project Management* [online]. Dostupné z: `http://www.brighthubpm.com/project-planning/110097-examples-of-using-storyboards-in-project-management/`, [cit. 19.04.2017]

[3] W3Schools Online Web Tutorials, *SVG Tutorial* [online]. Dostupné z: `https://www.w3schools.com/graphics/svg_intro.asp`, [cit. 19.04.2017]

[4] CHACON, Scott, STRAUB, Ben, *Pro Git* [online]. Dostupné z: `https://git-scm.com/book/en/v2`, [cit. 19.04.2017]

[5] vogella.com, *Eclipse RCP (Rich Client Platform) - Tutorial.* [online]. Dostupné z: `http://www.vogella.com/tutorials/EclipseRCP/article.html`, [cit. 19.04.2017]

[6] gradle.org, *Gradle User Guide Version 3.4.1.* [online]. Dostupné z: `https://docs.gradle.org/3.4.1/userguide/userguide.html`, [cit. 19.04.2017]

[7] GitHub - akhikhl/wuff, *Gradle plugin for automating assembly of OSGi/Eclipse bundles and applications.* [online]. Dostupné z: `https://github.com/akhikhl/wuff`, [cit. 19.04.2017]

[8] DUŽÍ, Marie a Pavel MATERNA., *TIL jako procedurální logika: průvodce zvídavého čtenáře Transparentní intensionální logikou.* Bratislava: Aleph, 2012. Noema, 7. sv. ISBN 978-80-89491-08-7.

A Uživatelská dokumentace

A.1 Popis aplikace

Aplikace je určena pro vytváření a editaci storyboardů. Umožňuje vytváření storyboardů z komponent (figure), které jsou již od začátku dostupné, případně umožňuje i vytvoření vlastních komponent, ze kterých lze seskládat jednotlivé scény storyboardu.

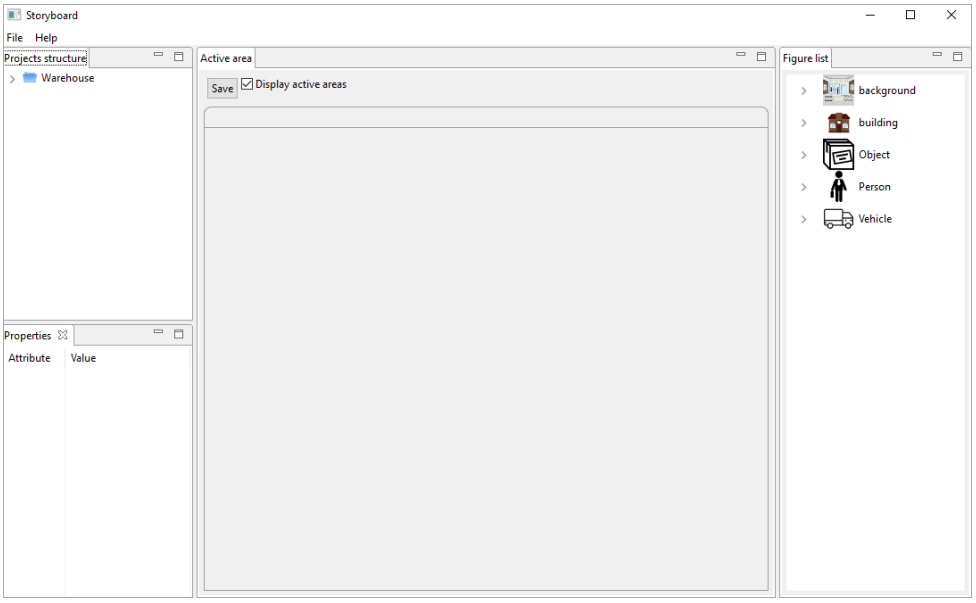
Aplikace byla vyvíjena jako semestrální projekt tříčlenné skupiny Ondřeje Horáčka, Václava Vaňka a Adama Zikmunda a následně vylepšována a rozšiřována jako diplomová práce Ondřeje Horáčka na VŠB - TU Ostrava v letech 2016-2017.

A.2 Spuštění aplikace

Aplikaci není potřeba instalovat, dodává se jako adresář, obsahující spustitelný soubor storyboard.exe, kterým se aplikace spouští, a další podadresáře obsahující soubory potřebné pro běh aplikace. Taky obsahuje složku sample-data, která obsahuje základní knihovnu figures a také složku projects, do které se ukládají projekty, vytvořené ve Storyboard editoru.

Jedná se o Java aplikaci, proto je potřeba mít běhové prostředí JRE.

Po prvním spuštění se zobrazí hlavní okno aplikace obsahující výchozí knihovnu již vytvořených figures a jeden ukázkový vytvořený projekt (obr. 31).



Obrázek 31: Hlavní okno aplikace

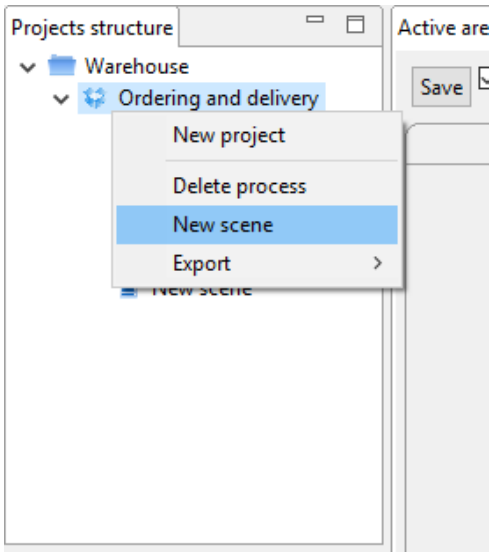
A.3 Vytváření projektů, procesů, scén

Pro vytváření storyboardů je potřeba mít vytvořený projekt. Ten lze vytvořit v části Projects structure. Kliknutím pravého tlačítka myši v této části se vyvolá nabídka menu a zvolením

možnosti New project se vytvoří nový projekt.

Aby bylo možné vytvářet storyboard, je potřeba do projektu přidat proces, který reprezentuje daný storyboard. Proces se přidá označením projektu, vyvoláním nabídky menu pravým tlačítkem myši a zvolení možnosti New process.

Procesy tedy reprezentuje celý storyboard, ovšem sestavení probíhá na jednotlivých scénách, proto je potřeba do procesu ještě přidat scény. Pro přidání scény je potřeba označit určitý proces a opět vyvolat nabídku menu pomocí pravého tlačítka myši. V tomto menu následně zvolit možnost New scene (obr. 32).

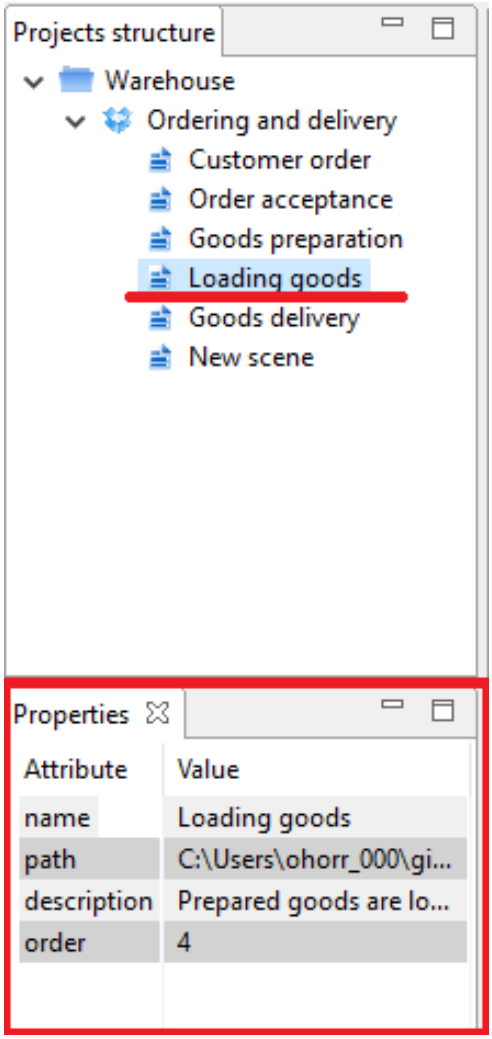


Obrázek 32: Vytváření nové scény

A.4 Změna vlastností projektů, procesů, scén

Projekty, procesy a scény obsahují při vytvoření výchozí názvy. Přejmenovávání se provádí v části Properties. Zvolením určitého objektu v části Projects structure se zpřístupní jeho vlastnosti v okně Properties.

V okně Properties je možné upravovat název projektu, procesu nebo scény a u scény dále umožňuje i nastavení popisu scény a pořadí dané scény v procesu (obr. 33).

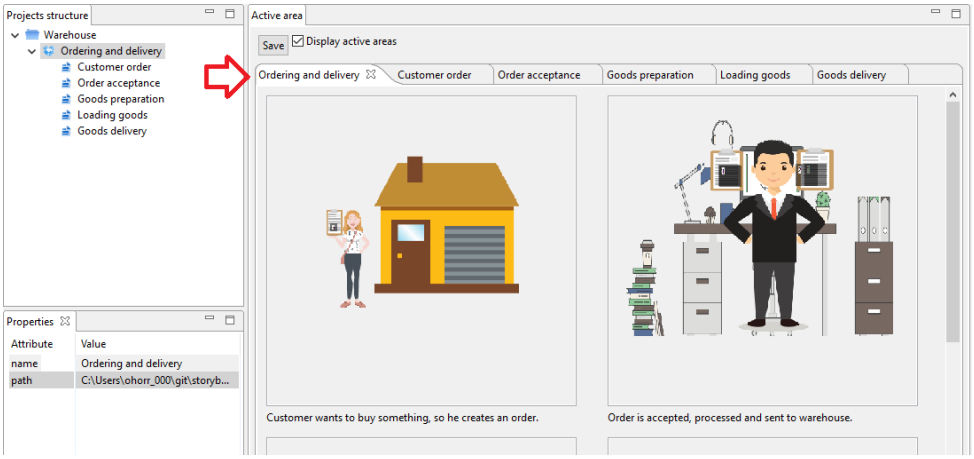


Obrázek 33: Zobrazení vlastností vybraného objektu

A.5 Sestavování scén

Sestavování scén se provádí v části Active area. Scénu, kterou chceme upravovat, otevřeme tak, že ji nalezneme v části Projects structure a načteme na část Active area pomocí dvojkliku levým tlačítkem myši na danou scénu. Tím se vytvoří v části Active area nová záložka obsahující danou scénu.

Scény lze otvírat buď po jedné dvojklikem na danou scénu v části Projects structure, nebo pomocí dvojkliku na celý proces, ve kterém je scéna obsažena. Tím se otevřou všechny scény daného procesu včetně záložky zobrazující přehled všech scén daného procesu (obr. 34).



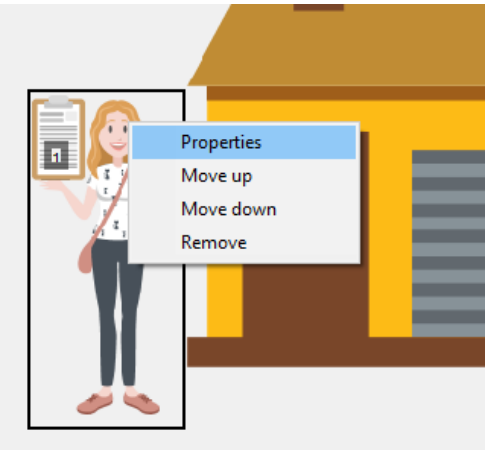
Obrázek 34: Načtení procesu na Active area

Po načtení scény lze přidávat na scénu figure. Figure se přidávají přetáhnutím figure z části Figure list na plochu Active area.

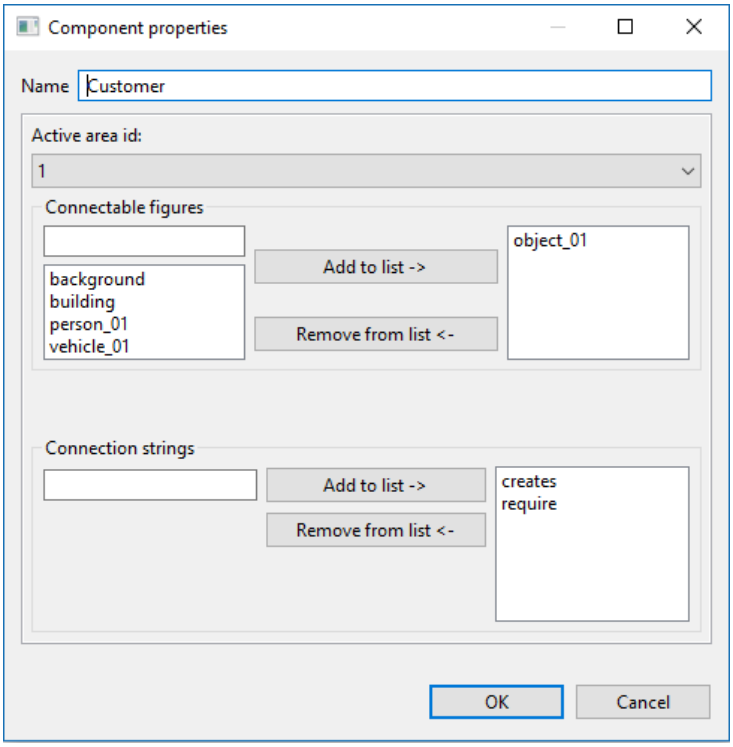
Pozor! Pro vykreslení figure na plochu Active area je potřeba, aby se celý figure dokázal vykreslit na plátno. Pozice kurzoru při přetahování figure z Figure listu určuje levý horní roh vykreslovaného figure. Pokud se figure nevejde na plátno celý, není vykreslen.

A.5.1 Změna vlastností figure na scéně

Pravým tlačítkem myši na daný figure načtený na scéně lze otevřít menu daného figure (obr. 35). Toto menu obsahuje 4 možnosti. První je možnost Properties, která otevře okno pro úpravu názvu figure a vlastností jednotlivých aktivních oblastí tohoto figure (obr. 36). Další možnosti v menu jsou Move up a Move down, které posunou objekt do popředí nebo do pozadí. Poslední možností je poté možnost Remove, která slouží pro smazání figure ze scény.



Obrázek 35: Menu pro figure umístěný na scéně



Obrázek 36: Okno pro úpravu figure umístěných na scéně - menu Properties

A.5.2 Změna pozice a rozměrů figure

Pro přesunutí figure na scéně slouží chycení objektu na scéně stisknutím a podržením levého tlačítka myši a přesunutí na požadovanou pozici.

Změna rozměrů se provádí pomocí dvojkliku na daný figure na scéně. Tím dojde k označení figure a následně chycením hrany figure dojde ke změně velikosti. Směr rozšiřování figure se určuje na základě prvního pohybu myši po chycení hrany figure.

Po označení figure je možné i posouvat jednotlivými aktivními oblastmi daného figure stejným způsobem, jako se přesunoval figure na scéně. Následně jde i označit aktivní oblast pomocí dvojkliku myši (obr. 37) a měnit její velikost stejným způsobem, jako se měnily rozměry figure.

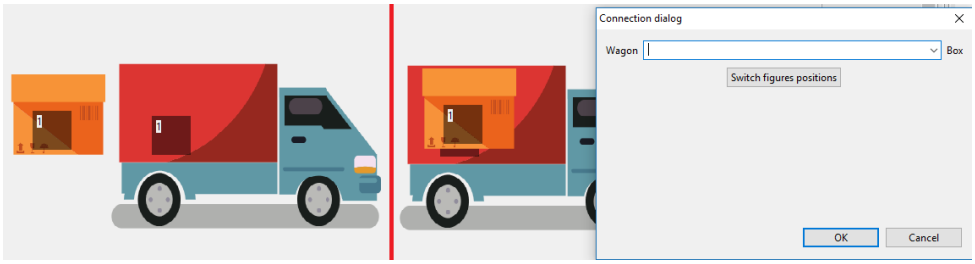
Pro odznačení zvoleného objektu stačí kliknout na nevyužitý prostor plátna.

Obrázek 37: Označení figure a následně i aktivní oblasti ve figure

A.5.3 Propojování figure na scéně

Na scéně jsou dvě možnosti, jak propojit figure.

První možnost je přesunout figure na scéně tak, aby se dané aktivní oblasti, které chceme propojit, alespoň částečně překrývaly (obr. 38). Pro vytvoření connection je potřeba, aby propojované aktivní oblasti měly v seznamu Connectable figures nastaven figure, ke kterému se snaží připojit, případně předka daného figure. Například, pokud mám figure s id figure1 a figure s id figure2, které chci propojit, musí mít aktivní oblast figure1 v seznamu Connectable figures uvedený figure2 nebo jeho předka a stejně tak musí mít aktivní oblast figure2 v seznamu Connectable figures uvedený figure1. Pokud je tato podmínka splněna, zobrazí se dialogové okno umožňující nastavení popisu pro dané propojení.



Obrázek 38: Vytvoření connection na scéně

Druhá možnost je poté taková, že má figure nastavenu vlastnost Background element. V takovém případě není potřeba propojovat dvě aktivní oblasti, nýbrž stačí propojit aktivní oblast tohoto figure přímo s jiným figure, není potřeba propojovat s další aktivní oblastí. Například, pokud máme figure1 s nastavenou vlastností Background element a figure2, stačí pro vytvoření propojení, aby měla aktivní oblast figure1 uvedený figure2 v seznamu Connectable figures a figure půjdou propojit, i když figure2 nebude mít žádnou aktivní oblast.

Vytvořený connection jde také následně přejmenovat zvolením možnosti Rename connection z kontextové nabídky menu při kliknutí pravým tlačítkem myši na aktivní oblast tvořící connection.

A.6 Vytváření a úprava figure

Vytváření a úprava figure se provádí v editoru (obr. 39. Editor lze otevřít z hlavní nabídky menu zvolením File -> Editor.

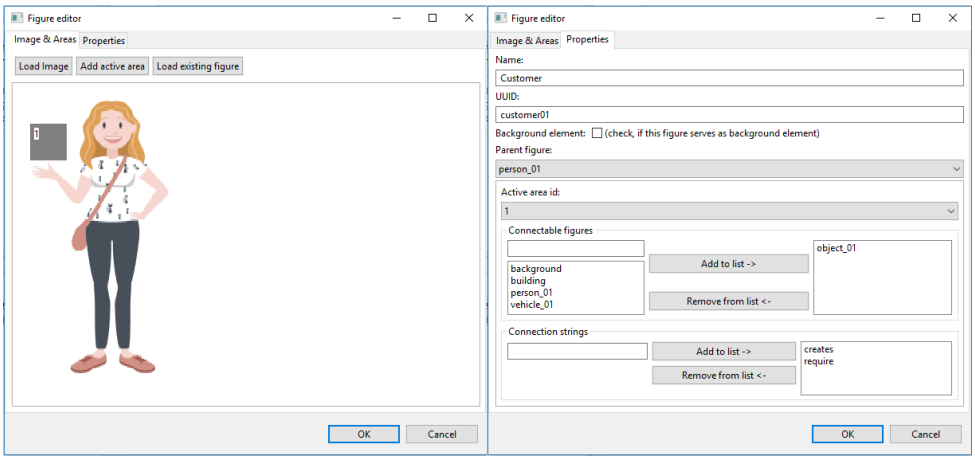
Pro vytvoření nového figure je potřeba v záložce `Image & Areas` zvolit možnost Load Image. Následně je potřeba nalézt SVG soubor, ze kterého bude figure tvořen. Po načtení SVG obrázku lze přidávat aktivní oblasti pomocí tlačítka Add active area. Pro načtení a úpravu již existujícího figure slouží možnost Load existing figure.

V záložce Properties lze následně nastavit vlastnosti figure. Mezi tyto vlastnosti patří:

- Name - Název figure
- UUID - Id figure, stejným názvem je pojmenovaný i soubor, do kterého se figure uloží
- Background element - Vlastnost, která umožňuje propojovat aktivní oblasti tohoto figure přímo s jinými figure bez nutnosti propojení přes aktivní oblasti
- Parent figure - Předek tohoto figure, slouží pro vytvoření hierarchie mezi figures

Dále je možné pro každou aktivní oblast nastavit další vlastnosti:

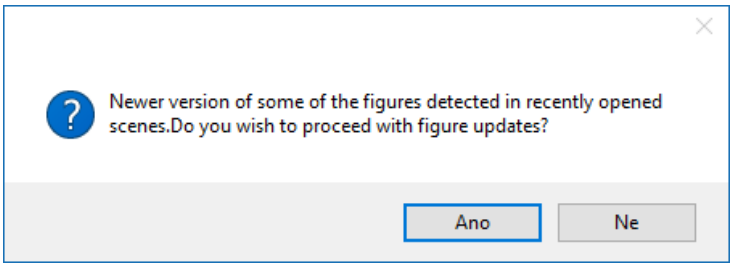
- Connectable figures - Figure v seznamu na pravé straně a taky potomky daného figure lze připojovat k dané aktivní oblasti. Levá strana obsahuje řádek pro vyhledávání specifických figure podle jejich id a také seznam nalezených figure. Pokud je řádek pro vyhledávání prázdný, obsahuje seznam všechny kořenové figure.
- Connection strings - Seznam textů nabízených pro popis connection při jejich vytváření.



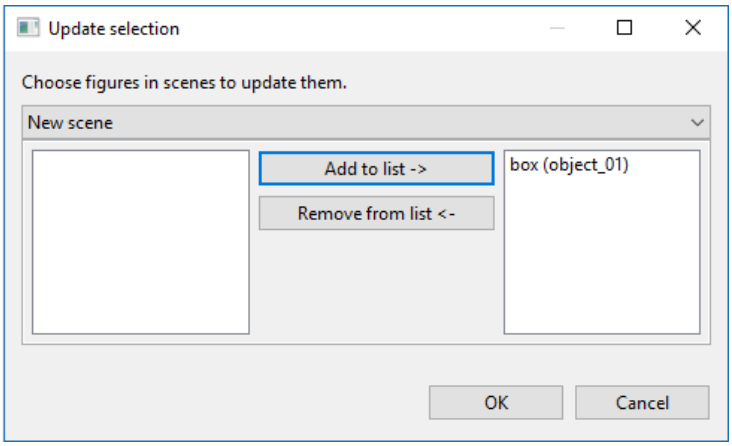
Obrázek 39: Editor pro vytváření a úpravu figure

A.7 Aktualizace figure

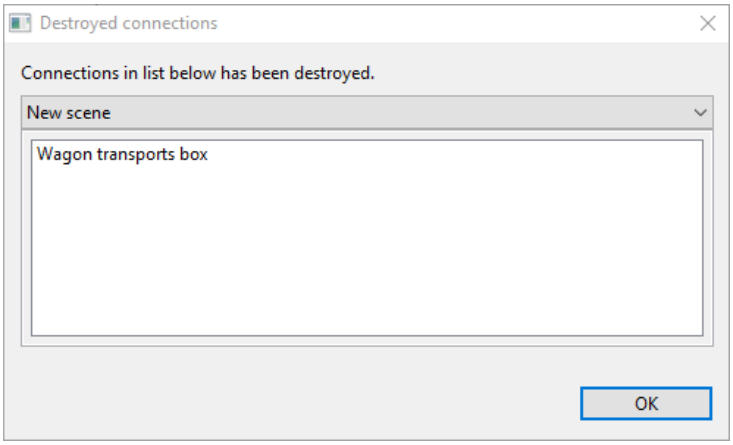
Pokud byl upraven figure, který je již využit v některé scéně, je při opětovném otevření dané scény zobrazen dialog, zda se má figure aktualizovat (obr. 40). Pokud je tato možnost potvrzena, zobrazí se dialog pro zvolení, které figure v dané scéně se mají aktualizovat (obr. 41). Seznam figure na levé straně obsahuje figure, které je možné aktualizovat. Seznam figure na pravé straně obsahuje potom ty figure, které budou aktualizovány po potvrzení dialogu. Pokud již aktualizovaný figure měl vytvořené nějaké connection, které by nebylo možné zachovat po aktualizaci figure, jsou tyto connection odebrány a následně je zobrazen seznam všech takto odebraných connection (obr. 42).



Obrázek 40: Dialog detekující změny figure



Obrázek 41: Dialog pro zvolení aktualizovaných figure

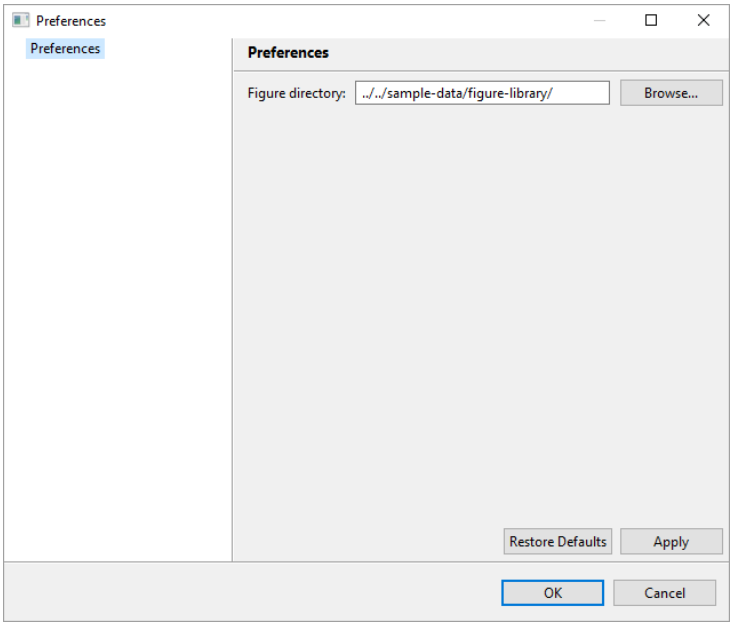


Obrázek 42: Dialog oznamující smazané connections

A.8 Změna adresáře figures

Aplikace poskytuje i možnost změny adresáře, ze kterého se načítají a do kterého se ukládají vytvořené figures. Výchozí umístění je nastaveno na adresář sample-data/figure-library/ ve složce aplikace.

Změna adresáře pro ukládání figures lze provést otevřením menu File -> Preferences. Tím se otevře okno, kde je možné změnit umístění tohoto adresáře (obr. 43).



Obrázek 43: Okno Preferences pro změnu umístění adresáře knihovny figures

B CD s aplikací

Přiložené CD obsahuje:

- Text této diplomové práce ve formátu PDF
- Složku project se zdrojovými kódy programu
- Složku storyboard se spustitelným souborem (Storyboard.exe)